Universitat de Girona

ADMISSION CONTROL SCHEMES FOR TCP ELASTIC TRAFFIC IN CLASS-BASED NETWORKS

Lluís FÀBREGA I SOLER

ISBN: 978-84-691-5960-6
Dipòsit legal: GI-1195-2008

Department of Computer Architecture and Technology

PhD Thesis

ADMISSION CONTROL SCHEMES FOR TCP ELASTIC TRAFFIC IN CLASS-BASED NETWORKS

Author: Lluís Fàbrega i Soler

Supervisor: Teodor Jové i Lagunas

Thesis presented in fulfillment of the requirements for the degree of PhD in Computer Engineering

Girona, April 2008

A la meva família.

Acknowledgements

I would like to thank here all the people that helped me throughout the last years to carry out this work.

Firstly, I am very grateful to my supervisor Dr. Teodor Jové, for his thoughtful guidance, encouragement and patience.

I would also like to thank all my colleagues in the Broadband Communications and Distributed System group for creating such a friendly working atmosphere. My gratitude to Anna, David, Juan, Eusebi, Fer, Sílvia, Lili, LuisFer and Joan, for their support and encouragement, and especially to Pere, Antonio, Josep Lluís and Ramon, for their help and advice so many times. Thanks also to Gemma, who as an undergraduate student helped me so much with the simulator.

I would also like to thank Dr. David Harle for accepting me as a research visitor at the University of Strathclyde in Glasgow, where I began the initial part of this work.

Finally, I thank the institutions that partially supported this work: the Spanish Science and Technology Ministry (under contracts TEC2006-03883, TIC2003-05567, TEL99-0976), the Generalitat of Catalonia's research support program (SGR 00296) and the University of Girona's research support fund (UdG-DinGruRec2003-GRCT40).

Abstract

The traditional "data" applications in the Internet, such as web browsing, peer-to-peer file sharing, ftp, e-mail and others, transfer discrete messages or "documents" (a web request, a basic web file, an embedded image, an ftp file, an ftp command, etc.). They are all built on top of TCP. Documents are partitioned into blocks and sent through the network into a sequence of packets or "flows" within TCP connections. The users of these applications expect that there is no error in the transfer of documents and also that the response time is the smallest possible below a certain maximum value. Absolute fidelity is achieved through TCP's retransmission procedures. This packet retransmission increases the packet delay and consequently the document transfer time. Moreover, it may cause duplicated packets, which are discarded by the destination. From the point of view of the network, the decisive Quality of Service (QoS) parameter is the average receiving rate or network throughput, which includes the duplicates. Since users expect the smallest possible response time below a certain maximum value, the network should provide the maximum possible throughput above a minimum value, a network service that we call the Minimum Throughput Service (MTS), and TCP should also be able to use it. Since the maximum possible throughput changes over time, TCP sources use rate-adaptive algorithms [jaco88a] that increase and decrease the sending rate in order to match these variations and minimize packet loss. "Data" applications and TCP flows are called "elastic" due to this ability to adjust the sending rate to different network conditions. Besides a variable average rate, elastic flows are very bursty. Finally, Internet traffic measurements show that the distribution of the document's size presents a heavy tail [crov97a], and as a consequence, most elastic flows are short and a few of them are very long.

The "traditional" network scheme on the Internet is based only on FIFO and Tail Drop queues, without traffic conditioning or Admission Control (AC) mechanisms. The combination of this scheme with TCP rate-adaptive algorithms aims to provide a throughput equal to the fair rate of the bottleneck link. Its main advantage is its simplicity. However, it is not able to provide different throughputs to different TCP flows. Moreover, it does not provide isolation between flows, i.e., flows sending at a higher rate than the fair throughput can damage other well-behaved flows. Finally, when resources in the followed path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied; otherwise, i.e., during congestion situations, none of them is satisfied.

An efficient way of dealing with congestion situations is to use an AC mechanism. With AC, when congestion occurs, some of the flows receive the minimum throughput (they are "accepted") and the rest do not receive it (they are "rejected" or "blocked"). Congestion situations can be reduced by increasing network resources or by optimizing its use through better routing techniques. However, if congestion still occurs, AC achieves an efficient use of network resources by maximizing the number of satisfied flows. The blocking rate depends on the behavior of users' demands, the chosen network provisioning, the routing techniques used, and the ability of the AC mechanism to maximize the number of satisfied flows. However, using an AC mechanism complicates the network scheme, and therefore a major concern is making the AC as simple as possible.

Our main objective is to design a network scheme with AC for TCP elastic traffic using simple mechanisms. Specifically, a network scheme that guarantees the MTS to the maximum

possible number of flows, where a flow is defined as a sequence of related packets within a TCP connection, that is able to provide different minimum throughputs to different users and isolation between flows, that is built with mechanisms (of traffic conditioning, queue disciplines and AC) that reduce the per-flow state, per-flow signaling and per-flow processing as much as possible, and that considers a multidomain scenario in which each domain has a user-provider agreement with each of its neighboring domains.

In this work we propose two network schemes with AC for TCP elastic traffic. Although the specific AC of each scheme is different, both schemes have a similar architecture. They are based on packet classes, with a different discarding priority assigned to each one. The AC is edge-to-edge and based on measurements, so that only edge routers participate in the AC and exchange signaling. The edge routers keep a list of active flows, which is used to detect new flows and to isolate the accepted flows by enforcing the agreed traffic profile to their input traffic. The AC is implicit: the start of a new flow is detected at the ingress when its first packet is received; the end of a flow is detected when no packet is received within a certain defined timeout interval; the request of the MTS and the desired minimum throughput are indicated through the port numbers and/or a specific mark in the packets in a way that has been specified in the user-provider agreement; acceptance is indicated simply by providing the MTS, while rejection is indicated by providing the best-effort service. The user-provider agreement also defines the value of the maximum aggregated throughput the user may ask for. Finally, the schemes also use pre-established logical paths from ingress points to egress points.

In our 1st scheme the AC is based on per-flow throughput measurements. The first packets of the flow are used as a probing flow. Flow's throughput is measured at the egress, and then signaling packets carry this measurement to the ingress, where the AC decision is made. In this scheme it is necessary to use a special modification of TCP sources. The short-term fluctuations of the sending rate of a "standard" TCP source reduce the performance, and to avoid this situation, we propose a modification of TCP's sending algorithms that keeps the short-term sending rate close to the actual average and above a minimum value.

Our 2nd scheme aims at achieving better performance for the "standard" TCP sources than our 1st scheme. The AC is based on per-aggregate throughput measurements. The out-profile packets of the aggregation of accepted flows (i.e., the packets that correspond to the flows' extra throughput), and in second term, a special flow (with packets marked as the highest discarding priority class), are used together as a probing flow. The aggregated throughput is measured at the egress, and then signaling packets carry this measurement to the ingress, where the AC decision is made.

We evaluate the performance of both schemes with simulations in several network topologies using different traffic loads consisting of TCP flows that carry files of varying sizes. We study the influence of several parameters on the performance of the schemes. The results confirm that our 1st scheme with the modified TCP and our 2nd scheme with the "standard" TCP guarantee the requested minimum throughput to the accepted flows and achieve good utilization of the network resources.

Resum

Les aplicacions de "dades" tradicionals a Internet, com la navegació web, la compartició de fitxers d'igual a igual, l'ftp, el correu electrònic i altres, transfereixen missatges discrets o "documents" (una petició web, un fitxer web bàsic, una imatge incrustada, un fitxer ftp, una comanda ftp, etc.) i estan totes construïdes sobre TCP. Els documents es parteixen en blocs i s'envien a través de la xarxa en seqüències de paquets o "fluxos" dins de connexions TCP. Els usuaris d'aquestes aplicacions esperen que no hi hagi cap error en la transferència dels documents i que el temps de resposta sigui el menor possible per sota d'un cert valor màxim. La fidelitat absoluta s'aconsegueix a través dels procediments de retransmissió de TCP. Aquesta retransmissió de paquets n'augmenta el retard i, en conseqüència, el temps de transferència del document, i a més, pot provocar paquets duplicats, els quals són descartats pel destí. Des del punt de vista de la xarxa, el paràmetre de qualitat de servei (QoS, *Quality of Service*) determinant és la velocitat mitjana de recepció o velocitat de transferència (*throughput*) de xarxa, el qual inclou els duplicats. Com que els usuaris esperen que el temps de resposta sigui el menor possible per sota d'un cert valor màxim, la xarxa ha de proporcionar el *throughput* màxim possible per sobre d'un valor mínim, un servei de xarxa que anomenem servei de *throughput* mínim (MTS, *Minimum Throughput Service*), i a més, TCP ha de ser capaç d'utilitzar-lo. Com que el *throughput* màxim possible canvia en el temps, les fonts TCP utilitzen algoritmes d'adaptació de velocitat [jaco88a] que augmenten i disminueixen la velocitat d'enviament per tal de seguir aquestes variacions i minimitzar la pèrdua de paquets. Per aquesta habilitat d'ajustar la velocitat d'enviament a les condicions de la xarxa, les aplicacions de "dades" i els fluxos TCP són anomenats "elàstics". A més d'una velocitat mitjana variable, el trànsit dels fluxos elàstics conté moltes ràfegues. Finalment, les mesures de trànsit a Internet mostren que la distribució de la grandària dels documents té una llarga cua [crov97a], i en conseqüència, la majoria dels fluxos elàstics són curts i uns quants són molt llargs.

L'esquema de xarxa "tradicional" a Internet està basat només en cues FIFO i *Tail Drop*, sense mecanismes de condicionament de trànsit ni de control d'admissió (AC, *Admission Control*). La combinació d'aquest esquema amb els algoritmes d'adaptació de velocitat de TCP té l'objectiu de proporcionar un *throughput* igual a la velocitat justa en l'enllaç més restrictiu ("coll d'ampolla"). El seu principal avantatge és la simplicitat. En canvi no és capaç de proporcionar *throughputs* diferents a fluxos TCP diferents. A més, no proporciona aïllament entre fluxos, és a dir, no evita que els fluxos que envien a una velocitat major que el *throughput* just facin mal als fluxos que es comporten bé. Finalment, quan els recursos en el camí seguit són suficients per satisfer els requeriments de *throughput* mínim de tots els fluxos, tots se satisfan, però en cas contrari, és a dir, durant les situacions de congestió, no se'n satisfà cap.

Una manera eficient de tractar les situacions de congestió és utilitzar un mecanisme d'AC. Amb AC, quan hi ha congestió, alguns dels fluxos reben el *throughput* mínim (són "acceptats") i la resta no el reben (són "rebutjats" o "bloquejats"). Les situacions de congestió es poden reduir amb un augment dels recursos de la xarxa o bé optimitzant-ne l'ús amb millors tècniques d'encaminament, però si tot i això hi ha congestió, l'AC aconsegueix un ús eficient dels recursos de la xarxa maximitzant el nombre de fluxos satisfets. El percentatge de bloqueig depèn del comportament de les demandes dels usuaris, el dimensionament de xarxa escollit, les tècniques d'encaminament utilitzades, i la capacitat del mecanisme d'AC de maximitzar el

nombre de fluxos satisfets. Amb tot, l'ús d'un mecanisme d'AC complica l'esquema de xarxa, i per tant la principal preocupació és fer-lo tan senzill com sigui possible.

El nostre principal objectiu és dissenyar un esquema de xarxa amb AC per al trànsit elàstic TCP utilitzant mecanismes senzills. Concretament, un esquema de xarxa que garanteixi l'MTS al màxim nombre possible de fluxos, on un flux es defineix com una seqüència de paquets relacionats dins d'una connexió TCP, que sigui capaç de proporcionar *throughputs* mínims diferents a usuaris diferents i aïllament entre fluxos, que estigui construït amb mecanismes (de condicionament de trànsit, de disciplines de cues i d'AC) que redueixin l'estat per flux, la senyalització per flux i el processament per flux, tant com sigui possible, i que consideri un escenari multidomini on cada domini tingui un acord usuari-proveïdor amb cadascun dels seus dominis veïns.

En aquest treball proposem dos esquemes de xarxa amb AC per al trànsit elàstic TCP. Encara que l'AC concret de cada esquema és diferent, ambdós esquemes tenen una arquitectura similar. Estan basats en classes de paquets, on cadascuna té assignada una prioritat de descart diferent. L'AC és extrem a extrem i basat en mesures, de manera que només els nodes frontera participen en l'AC, intercanvien senyalització i mantenen una llista de fluxos actius, la qual és utilitzada per detectar nous fluxos i per aïllar entre si els fluxos acceptats, forçant el perfil de trànsit acordat al seu trànsit d'entrada. L'AC és implícit: l'inici d'un nou flux es detecta a l'entrada quan se'n rep el primer paquet; la fi d'un flux es detecta quan no es rep cap paquet durant un interval de temps definit; la petició de l'MTS i del *throughput* mínim desitjat s'indiquen a través del número de port i/o una marca concreta en els paquets, d'una manera especificada en l'acord usuari-proveïdor; l'acceptació del flux s'indica senzillament proporcionant l'MTS, i el rebuig proporcionant el servei tant-com-puc (*best-effort*). L'acord usuari-proveïdor també defineix el valor del *throughput* agregat màxim que l'usuari pot demanar. Finalment, els esquemes també utilitzen camins lògics preestablerts des de punts d'entrada a punts de sortida.

En el nostre primer esquema l'AC està basat en mesures de *throughput* per flux. Els primers paquets del flux s'utilitzen com a flux sonda. El *throughput* del flux es mesura a la sortida, i llavors paquets de senyalització porten aquesta mesura a l'entrada, on es pren la decisió d'AC. L'esquema requereix l'ús d'una modificació especial de les fonts TCP. Les fluctuacions a curt termini de la velocitat d'enviament d'una font TCP "estàndard" redueixen el rendiment, i per evitar-ho, proposem una modificació dels algoritmes d'enviament de TCP que manté la velocitat d'enviament a curt termini a prop del valor mitjà actual i per sobre d'una velocitat mínima.

El nostre segon esquema té l'objectiu d'aconseguir un rendiment millor amb les fonts TCP "estàndard" que el primer esquema. L'AC està basat en mesures de *throughput* per agregats. Els paquets fora de perfil de l'agregació de fluxos acceptats (és a dir, els paquets que corresponen al *throughput* extra dels fluxos), i en segon terme, un flux especial (amb paquets marcats amb la classe de prioritat de descart més alta), són utilitzats conjuntament com a flux sonda. El *throughput* agregat es mesura a la sortida, i llavors paquets de senyalització porten aquesta mesura a l'entrada, on es pren la decisió d'AC.

Avaluem el rendiment d'ambdós esquemes a través de simulació en diverses topologies de xarxa utilitzant diferents càrregues de trànsit formades per fluxos TCP que porten fitxers de grandària variable. Estudiem la influència de diversos paràmetres en el rendiment dels esquemes. Els resultats confirmen que el primer esquema amb el TCP modificat i el segon, amb el TCP "estàndard", garanteixen el *throughput* mínim demanat als fluxos acceptats i aconsegueixen una bona utilització dels recursos.

Contents

Acknowledgements v

Abstract vii

Resum ix

Contents xi

List of figures xv

List of acronyms xix

Chapter 1: Introduction 1
1.1 Motivation 1
1.2 Objectives 3
1.3 Outline of the document 4

Chapter 2: Quality of Service (QoS) and TCP elastic traffic in the Internet 7
2.1 An introduction to multiservice packet networks 7
2.1.1 The concept of multiservice packet network 8
2.1.2 Application QoS, system QoS and network QoS 8
2.1.3 The QoS of real-time and "data" applications 12
2.1.4 Service guarantees 14
2.1.5 Service Level Agreements (SLAs) 15
2.1.6 Interconnection in a multidomain network 16
2.2 An overview of the mechanisms for building network services 16
2.2.1 Delay and loss in packet switching networks 16
2.2.2 Network mechanisms for service building 17
2.2.3 Service guarantees and network mechanisms 20
2.2.4 Core-stateful networks versus core-stateless networks. Class-based networks 22
2.3 Multiservice network architectures in the Internet 24
2.3.1 The "traditional" Internet 24
2.3.2 Integrated Services (Intserv) 25
2.3.3 Differentiated Services (Diffserv) 27
2.3.4 MultiProtocol Label Switching (MPLS) 30
2.3.5 Intserv, Diffserv and MPLS: a comparison 32
2.4 TCP elastic traffic 33
2.4.1 QoS for elastic traffic 33

2.4.2 The Minimum Throughput Service (MTS) 35
2.4.3 Reliability and resource sharing in TCP 35
2.4.4 Characteristics of TCP elastic traffic 41

Chapter 3: Admission Control (AC) schemes in the Internet 45
3.1 The AC mechanism 45
3.1.1 Introduction to AC 45
3.1.2 A classification of AC schemes 47
3.1.3 The AC algorithm, the reservations and the signaling 47
3.2 Centralized AC schemes 50
3.3 Distributed hop-by-hop AC schemes 51
3.3.1 Classical parameter-based hop-by-hop schemes 52
3.3.2 Classical measurement-based hop-by-hop schemes 53
3.3.3 Lightweight hop-by-hop schemes 55
3.4 Distributed one-hop AC schemes on logical paths (LPs) with reservation 58
3.5 Distributed edge-to-edge AC schemes 59
3.5.1 Active measurement-based edge-to-edge schemes with per-flow probing 60
3.5.2 Active measurement-based edge-to-edge schemes with per-aggregate probing 61
3.5.3 Passive measurement-based edge-to-edge schemes 62
3.6 Conclusions 63

Chapter 4: Network schemes for TCP elastic traffic proposed in the Internet 65
4.1 Network schemes for TCP elastic traffic without AC 65
4.1.1 The “traditional” scheme 66
4.1.2 Schemes based on packet classes 67
4.1.3 Core-Stateless Fair Queuing (CSFQ) based scheme 71
4.1.4 User-Share Differentiation (USD) scheme 72
4.2 Network schemes for TCP elastic traffic with AC 72
4.2.1 The scheme for a throughput service in Corelite 73
4.2.2 Implicit AC for TCP connections 75
4.2.3 The scheme for elastic traffic in Cross-Protect 76
4.3 Conclusions 77

Chapter 5: The 1st scheme: a network scheme for TCP elastic traffic with AC using edge-to-edge per-flow measurements in class-based networks 81
5.1 Introduction 81
5.2 Description of the scheme 83
5.2.1 The architecture 84
5.2.2 The influence of the short-term fluctuations of the TCP sending rate 85
5.2.3 The modification of the TCP source 86
5.2.4 Interdomain aspects and edge node operations 86
5.3 Evaluation of the scheme 88
5.3.1 Scalability study 88

5.3.2 Description of the simulation 89
5.3.3 Simulation results 92
5.4 Conclusions 104

Chapter 6: The 2nd scheme: a network scheme for TCP elastic traffic with AC using edge-to-edge per-aggregate measurements in class-based networks 107
6.1 Introduction 107
6.2 Description of the scheme 109
6.2.1 The architecture 109
6.2.2 Interdomain aspects and edge node operations 111
6.3 Evaluation of the scheme 112
6.3.1 Scalability study 112
6.3.2 Description of the simulation 113
6.3.3 Simulation results 114
6.4 Conclusions 124

Chapter 7: Conclusions and future work 127
7.1 Conclusions 127
7.2 Future work 129

Bibliography 131

Appendix A: Details of the simulation in ns-2 141

Appendix B: Related author's publications 145

List of figures

FIGURE 2-1: Application QoS, system QoS and network QoS. 9

FIGURE 2-2: Traffic parameters and QoS parameters of a packet flow. 11

FIGURE 2-3: Queuing delays and losses experienced by a flow in a packet switching network. 17

FIGURE 2-4: The layers of the TCP/IP architecture. 24

FIGURE 2-5: The traditional protocols in the TCP/IP architecture. 25

FIGURE 2-6: Functional block diagram of the Integrated Services architecture. 27

FIGURE 2-7: Functional block diagram of the Differentiated Services architecture. 28

FIGURE 2-8: The basics of the MPLS architecture. 31

FIGURE 2-9: Application QoS and network QoS in elastic applications. 34

FIGURE 2-10: The definition of the Minimum Throughput Service. 35

FIGURE 2-11: The ideal AIMD (Additive Increase Multiplicative Decrease) behavior of the TCP congestion window, after an initial Slow Start phase. 39

FIGURE 3-1: The AC evaluates if the service requested by a new flow can be provided while maintaining the service promised to the actual flows in the path. 46

FIGURE 3-2: Classification of AC schemes. 48

FIGURE 4-1: Dropping probability as a function of the queue's average occupancy in RED. 67

FIGURE 4-2: Dropping probability as a function of the queue's average occupancy in RIO. 69

FIGURE 5-1: Functional block diagram of our 1st scheme. 85

FIGURE 5-2: Interdomain operation of a domain using our 1st scheme with neighboring domains. 87

FIGURE 5-3: Network Topologies 1, 2, 3, 4 and 5, indicating the link's capacity and propagation delay, and the length of queues. 90

FIGURE 5-4: Comparison of our 1st scheme with the modified TCP (MO), with TCP NewReno (NR), with TCP Westwood (WE) and with TCP SACK (SA), and the "traditional" scheme (best-effort service) with TCP NewReno (NR), in Topology 1 ($Tfl_{03} = Tfl_{23} = 0.2$ s). 93

FIGURE 5-5: Sample mean (m) and 95% confidence intervals (ci) of the performance parameters obtained with our 1st scheme and the modified TCP in Topology 1 ($Tfl_{03} = Tfl_{23} = 0.2$ s). 94

FIGURE 5-6: For our 1st scheme and the modified TCP in Topology 1 ($Tfl_{03} = Tfl_{23} = 0.2$ s), at the top, percentage of satisfied flows and non-satisfied flows over the accepted flows versus the offered traffic load, and at the bottom, frequency distribution of throughput of accepted flows when the offered traffic load of each logical path is 3 Mbps (in total, 6 Mbps). 95

FIGURE 5-7: For the "traditional" scheme (best-effort service) and TCP NewReno in Topology 1, percentages over all flows of "completed" and "achieved" flows, versus the offered traffic load.....96

FIGURE 5-8: The influence of the measurement duration Tfl_{LP} on the performance of our 1st scheme and the modified TCP, in Topology 1 ($Tfl_{03} = Tfl_{23}$).....97

FIGURE 5-9: For our 1st scheme and the modified TCP, results for two users asking for different minimum throughputs, 90 Kbps and 135 Kbps, in Topology 1 ($Tfl_{03} = Tfl_{23} = 0.2$ s).....98

FIGURE 5-10: For our 1st scheme and the modified TCP, isolation of TCP flows against CBR flows of 135 Kbps, when the desired minimum throughput of TCP and CBR flows is 90 Kbps, in Topology 1 ($Tfl_{03} = Tfl_{23} = 0.2$ s).....99

FIGURE 5-11: The influence of the packet RTT on the performance of our 1st scheme with the modified TCP, in Topology 2 ($Tfl_{03} = Tfl_{23} = 0.2$ s).....100

FIGURE 5-12: For our 1st scheme and the modified TCP, sharing between logical paths in Topology 1 ($Tfl_{03} = Tfl_{23} = 0.2$ s). The offered traffic load of e0e3 is constant and equal to 1 Mbps.....101

FIGURE 5-13: The influence of the number of hops on the performance of our 1st scheme with the modified TCP, in Topology 3 ($Tfl_{08} = 0.4$ s, $Tfl_{15} = Tfl_{37} = 0.2$ s).....102

FIGURE 5-14: The influence of the number of hops on the performance of our 1st scheme with the modified TCP, in Topology 4 ($Tfl_{06} = Tfl_{26} = 0.4$ s, $Tfl_{36} = Tfl_{56} = 0.2$ s).....103

FIGURE 5-15: The influence of the number of hops on the performance of our 1st scheme with the modified TCP, in Topology 5 ($Tfl_{09} = Tfl_{29} = 0.5$ s, $Tfl_{39} = Tfl_{59} = 0.4$ s, $Tfl_{69} = Tfl_{89} = 0.2$ s).....104

FIGURE 6-1: Functional block diagram of our 2nd scheme.....109

FIGURE 6-2: Obtaining the available throughput in a path Rav_{LP} from the measurement Mag_{LP}.....111

FIGURE 6-3: Interdomain operation of a domain using our 2nd scheme with neighboring domains.....111

FIGURE 6-4: Comparison of our 2nd scheme, our 1st scheme and the "traditional" scheme (best-effort service), using TCP NewReno in Topology 1 ($Tag_{03} = Tag_{23} = 0.5$ s, $Tfl_{03} = Tfl_{23} = 0.5$ s).....115

FIGURE 6-5: For our 2nd scheme and TCP NewReno in Topology 1 ($Tag_{03} = Tag_{23} = 0.5$ s), at the top, percentage of satisfied flows and non-satisfied flows over the accepted flows versus the offered traffic load, and at the bottom, frequency distribution of throughput of accepted flows when the offered traffic load of each logical path is 3 Mbps (in total, 6 Mbps).....116

FIGURE 6-6: The influence of the measurement duration Tag_{LP} on the performance of our 2nd scheme and TCP NewReno, in Topology 1 ($Tag_{03} = Tag_{23}$).....117

FIGURE 6-7: For our 2nd scheme and TCP NewReno, results for two users asking for different minimum throughputs, 90 Kbps for user A and 135 Kbps for user B, in Topology 1 ($Tag_{03} = Tag_{23} = 0.5$ s).....119

FIGURE 6-8: For our 2nd scheme and TCP NewReno, isolation of TCP flows against CBR flows of 135 Kbps, when the desired minimum throughput of TCP and CBR flows is 90 Kbps, in Topology 1 ($Tag_{03} = Tag_{23} = 0.5$ s).....120

FIGURE 6-9: The influence of the packet RTT on the performance of our 2nd scheme and TCP NewReno, in Topology 2 ($Tag_{03} = Tag_{23} = 0.5$ s).....121

FIGURE 6-10: For our 2nd scheme and TCP NewReno, sharing between logical paths in Topology 1 (Tag_{03} = Tag_{23} = 0.5 s). The offered traffic load of e0e3 is constant and equal to 1 Mbps. 122

FIGURE 6-11: The influence of the number of hops on the performance of our 2nd scheme and TCP NewReno, in Topology 3 (Tag_{08} = Tag_{15} = Tag_{37} = 0.5 s). 123

FIGURE 6-12: The influence of the number of hops on the performance of our 2nd scheme and TCP NewReno, in Topology 4 (Tag_{06} = Tag_{26} = Tag_{36} = Tag_{56} = 0.5 s)..... 124

FIGURE 6-13: The influence of the number of hops on the performance of our 2nd scheme and TCP NewReno, in Topology 5 (Tag_{09} = Tag_{29} = Tag_{39} = Tag_{59} = Tag_{69} = Tag_{89} = 0.5 s). 125

List of acronyms

AC	Admission Control
ACK	Acknowledgment
AF	Assured Forwarding
AIMD	Additive Increase and Multiplicative Decrease
AS	Autonomous System
ATM	Asynchronous Transfer Mode
Bag_{LP}	Number of bits of aggregates measured in Tag_{LP} in logical path LP (2nd scheme)
Bfl_{LP}^{f}	Number of bits of flow f measured in Tfl_{LP} in logical path LP (1st scheme)
BA	Behavior Aggregate
BB	Bandwidth Broker
CBDP	Class-Based Discarding Priority
CBR	Constant Bit Rate
CSFQ	Core-Stateless Fair Queuing
cwnd	Congestion window
Diffserv	Differentiated Services
DPS	Dynamic Packet State
DS	Differentiated Services
DSCP	DS (Differentiated Services) Code Point
ECN	Explicit Congestion Notification
EF	Expedited Forwarding
EMBAC	End-to-end Measurement-Based AC
FEC	Forwarding Equivalent Class
FIFO	First-In-First-Out
FQ	Fair Queuing
IETF	Internet Engineering Task Force
Intserv	Integrated Services
IP	Internet Protocol
IS	Integrated Services
ISPN	Integrated Service Packet Network
LP	Logical Path
LSP	Label Switched Path
LSR	Label Switching Router
Mag_{LP}	Throughput measurement of aggregates in logical path LP (2nd scheme)

Mfl_{LP}^{f}	Throughput measurement of flow *f* in logical path *LP* (1st scheme)
MBAC	Measurement-Based Admission Control
MF	Multifield
MPLS	MultiProtocol Label Switching
MSS	Maximum Segment Size
MTS	Minimum Throughput Service
PBAC	Parameter-Based Admission Control
PDB	Per-Domain Behavior
PFQ	Priority Fair Queuing
PHB	Per-Hop-Behavior
QoS	Quality of Service
$Racc_{LP}^{f}$	Minimum throughput of an accepted flow *f* in logical path *LP* (2nd scheme)
Rav_{LP}	Available throughput in logical path *LP* (2nd scheme)
$Rreq_{LP}^{f}$	Minimum throughput of a requesting flow *f* in logical path *LP*
RED	Random Early Detection
RFC	Request For Comments
RIO	RED (Random Early Detection) with In and Out bits
RSVP	Resource ReSerVation Protocol
RTO	Retransmit Timeout
RTCP	RTP Control Protocol
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
SACK	Selective ACK (Acknowledgment)
SLA	Service Level Agreement
SLS	Service Level Specification
SNMP	Simple Network Management Protocol
SRP	Scalable resource Reservation Protocol
ssthresh	Slow Start threshold
Tag_{LP}	Duration of measurement period of aggregates in logical path *LP* (2nd scheme)
Tfl_{LP}	Duration of measurement period of a flow in logical path *LP* (1st scheme)
TCP	Transmission Control Protocol
TSW	Time-Sliding Window
UDP	User Datagram Protocol
WFQ	Weighted Fair Queuing

Chapter 1:

Introduction

This chapter serves as an introduction to the rest of the document. We present the motivation for this research work as well as the desired objectives, and then we describe the structure and the contents of this document.

1.1 Motivation

In the last decades a lot of effort has been put into making the Internet evolve so that it supports any kind of application satisfactorily. The new network architecture should be able to provide the different services required by different applications and also the assurances that users require. Moreover, since the Internet is composed of many networks or domains, the aspects of this multidomain context should also be taken into account. In this work we focus on network schemes that provide the service required by the traditional Internet applications, such as web browsing, peer-to-peer file sharing, ftp, e-mail and others, which generate the majority of Internet traffic. All these applications are built on top of TCP and are known as "data" applications.

In "data" applications, application's processes transfer discrete (time-independent) messages or "documents" (a web request, a basic web file, an embedded image, an ftp file, an ftp command, etc.). The performance of these applications is described in terms of fidelity to the original documents and in terms of interactivity or response time. Fidelity refers to the errors in the transferred documents, and since users expect no errors, absolute fidelity is required. The definition of the response time varies per application (e.g., in web browsing it is the waiting time between a page request and its visualization) and is composed of the transfer times of several documents and the processing time of the application's processes (e.g., a web server). The smaller the response time the more satisfied the user, but when the response time is too long, impatient users or high layer protocols may interrupt the transfer, which implies a waste of resources that can get even worse if the transfer is tried again [mass99a]. This means there is a maximum response time, which depends on users' desires (some may be more demanding than others) and the specific application. In conclusion, the users of these

applications expect that there is no error in the transfer of documents and also that the response time is the smallest possible below a certain maximum value.

For the transfer of an individual document, the above requirements imply that there should be no error and the smallest possible transfer time below a certain maximum value. Absolute fidelity in the document transfer is achieved through TCP's packet retransmission procedures. Each document is partitioned into blocks and sent into a sequence of packets within a TCP connection. From the acknowledgment packets sent back by the destination, the source detects and retransmits the lost packets until the whole document is received correctly. Packet retransmission increases the packet delay and consequently the document transfer time. Moreover, it may cause duplicated packets, which are discarded by the destination. From the point of view of the network, the decisive Quality of Service (QoS) parameter is the average receiving rate or network throughput, which includes the duplicates (from the point of view of the application, the average receiving rate without including the duplicates, known as goodput, is more important). The requirement concerning the document transfer time turns into a requirement concerning the throughput, i.e., the document transfer should achieve the maximum possible throughput above a minimum value. This implies that the network should provide a minimum throughput and if possible, an extra throughput, a network service that we call the Minimum Throughput Service (MTS), and also that TCP should be able to use it. TCP sources use rate-adaptive algorithms to achieve the maximum possible throughput while sharing the network resources fairly between TCP flows [jaco88a]. Since the maximum possible throughput changes over time, TCP increases and decreases the sending rate in order to match these variations and minimize packet loss. Due to this ability to adjust the sending rate to different network conditions, "data" applications and TCP flows are called elastic.

It is important to bear in mind several traffic characteristics of TCP elastic flows. Firstly, determining a TCP flow in the network may be difficult because its relation with TCP connections is not obvious. A flow is a sequence of "related" packets that are "close" in time, which can correspond to the transfer of a single document, several documents or an entire application session. It can correspond to a sequence of packets within a single TCP connection, within several connections or an entire TCP connection. A flow can be identified by the usual 5-tuple in IPv4, i.e., protocol, source and destination IP addresses and ports (or, if possible, by the more flexible 3-tuple in IPv6, flow label, source and destination IP addresses), initiated when the first packet arrives and finished when there are no more packets during a timeout period, although another option is to equate a flow to an entire TCP connection. Secondly, the sequence of packets corresponding to a document transfer (which includes the retransmitted packets) is typically very bursty (the TCP source sends a number of packets continuously – a burst – according to the actual window and then stops and waits for acknowledgments before going on) and has a variable average rate (due to the TCP rate-adaptive algorithms). Moreover, besides data packets, control packets for connection management and error correction are also sent. Finally, another important feature of TCP traffic, which has been observed in Internet traffic measurements, is that the distribution of the document's size has a heavy tail [crov97a]. This means that there is a high variability in sizes, most of the documents are small and a few of them are very large. As a consequence, most of elastic flows are short and a few of them are very long.

The "traditional" network scheme in the Internet is based only on FIFO and Tail Drop queues, and traffic conditioning or Admission Control (AC) mechanisms are not used. The combination of this scheme with TCP rate-adaptive algorithms aims at providing a throughput equal to the fair rate of the bottleneck link (i.e., the link's capacity divided by the number of present flows), although the effective resource sharing may exhibit unfairness in some situations. The main advantage of this scheme is the simplicity of the network mechanisms. However, it is not able to provide different throughputs to different TCP flows. Moreover, it does not provide isolation between flows, i.e., flows sending at a higher rate than the fair throughput can damage other

well-behaved flows. In consequence, the scheme relies on the cooperation between TCP sources that implement the same algorithms, but if some sources do not adapt the sending rate or adapt it in a different way, well-behaved TCP flows receive smaller throughput. Finally, when resources in the followed network path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied, but otherwise, i.e., during congestion situations, none of them is satisfied. Congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques. If the network resources are over-provisioned so that congestion never or rarely occurs, this scheme always provides the desired minimum throughput to all flows.

Over-provisioning the network resources is a common practice in backbone networks, since it allows simple network schemes to be used. However, over-provisioning can be difficult to achieve since unexpected events may happen (inaccurate traffic forecasts, routing changes, link or router failures, etc.), and it can be very inefficient in using resources. If more efficient provisioning is desired, another possible option is using schemes that include an AC mechanism. By using AC, when resources in the followed path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied, and otherwise, i.e., during congestion situations, some of them receive the minimum throughput (they are "accepted") and the rest do not receive it (they are "rejected" or "blocked"). Again, congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques, but if congestion still occurs, AC achieves efficient use of resources by maximizing the number of satisfied flows. The blocking rate depends on the behavior of users' demands, the chosen network provisioning, the routing techniques used, and the ability of the AC mechanism to maximize the number of satisfied flows. However, using AC complicates the network scheme, and therefore a major concern is making the AC as simple as possible. Although AC has been extensively studied in the literature, most of these studies deal with real-time traffic and only a few of them with TCP elastic traffic. Therefore, AC for TCP elastic traffic is an open research topic that has motivated this work.

1.2 Objectives

Our main objective is to design a network scheme with AC for TCP elastic traffic using simple mechanisms. Specifically, the general requirements for the desired network scheme are the following:

- It should guarantee the MTS to the maximum possible number of flows, that is, a minimum throughput and if possible, an extra throughput that comes from sharing the remaining resources between competing flows.
- It should consider that a flow is a sequence of related packets within a TCP connection.
- It should be able to provide different minimum throughputs to different users.
- It should provide isolation between flows (so that flows sending more traffic than their allocated throughput do not damage well-behaved flows that do send according to their allocated throughput).
- It should be built with simple mechanisms, i.e., with mechanisms (of traffic conditioning, queue disciplines and AC) that reduce the per-flow state, per-flow signaling and per-flow processing as much as possible.
- It should consider a multidomain scenario, where each domain has a user-provider agreement (SLA) with each of its neighboring domains, and the scheme is used in one of the domains.

In order to reach the main objective of this work we have determined the following set of steps or subobjectives:

- To study the basic concepts of multiservice network architectures, the definition and building of services through network mechanisms (especially the role of AC) and the characteristics of TCP elastic traffic (the network service it requires and several aspects of TCP).
- To study the main AC schemes that have been proposed in the Internet, focusing on their simplicity regarding how many nodes participate in the AC, the required state, the use of signaling, etc.
- To study the main network schemes that have been proposed in the Internet for TCP elastic traffic, with or without AC, focusing on the main characteristics of the service (whether the minimum throughput can be different or is the same for all flows, whether isolation between flows is provided, etc.) and their architecture (the specific traffic conditioning, queue disciplines and AC mechanisms used, the required state, the use of signaling, etc.).
- To propose new network schemes with AC for TCP elastic traffic using simple mechanisms.
- To evaluate the proposal through simulation. This will require implementing the scheme in a simulator and choosing the appropriate tests to determine whether it accomplishes the desired requirements.

1.3 Outline of the document

This document is organized into 7 chapters including this one, plus the bibliography and two appendixes at the end.

In Chapter 2 we first review the main aspects of multiservice packet networks, including the concepts of Quality of Service (QoS) and service guarantees. Then we describe the set of network mechanisms that can be used for building network services. We then present the main multiservice network architectures that have been proposed in the Internet. Next we deal with TCP elastic traffic, its characteristics and QoS requirements, since our work is centered on this traffic.

In Chapter 3 we study the main AC schemes that have been proposed in the Internet. Firstly we deal with general issues of the AC mechanism and classify the AC schemes into centralized, hop-by-hop, one-hop on LPs with reservation and edge-to-edge schemes. Then we describe specific AC schemes in the Internet according to this classification.

In Chapter 4 we review the main network schemes that have been proposed in the Internet for TCP elastic traffic. We describe these network schemes classified into two broad groups, the ones that do not use AC and the ones that do use it. For each of them we describe the main characteristics of the service provided and its architecture.

In Chapter 5 we describe our first proposal of a network scheme with AC for TCP elastic traffic. The whole scheme uses a small set of packet classes, with a different discarding priority assigned to each one. The AC is implicit, edge-to-edge and based on per-flow throughput measurements. However, the short-term fluctuations of the sending rate of a "standard" TCP source reduce the performance, and to avoid this situation, we propose a modification of TCP's sending algorithms. The chapter is organized as follows: firstly we present the basic features of the scheme and we discuss the reasons behind its architecture; then we describe the scheme in detail including the modification of the TCP source; and finally, we show the simulation results we have obtained to evaluate the performance.

In Chapter 6 we describe our second proposal of a network scheme with AC for TCP elastic traffic. The main goal is to achieve better performance for "standard" TCP flows since our 1st scheme requires a special modification of TCP's sending algorithms. The whole scheme also uses a small set of packet classes, with a different discarding priority assigned to each one.

The AC is implicit, edge-to-edge and based on per-aggregate throughput measurements. The chapter is organized as follows: firstly we present the basic features of the scheme and we discuss the reasons behind its architecture; then we describe the scheme in detail; finally, we show the simulation results we have obtained to evaluate the performance.

In Chapter 7 we summarize the main contributions of this work and point out possible directions for future research.

At the end, in appendix A we describe several details of the implementation of both schemes in the ns-2 simulator [ns-2]. In appendix B we list the international journals, proceedings of international conferences and research reports where this work has been published.

Chapter 2:

Quality of Service (QoS) and TCP elastic traffic in the Internet

The explosive growth and success of the Internet in the last decades is causing the evolution of its network architecture. The Internet is expected to support any kind of application satisfactorily, from the traditional "data" applications to the new real-time applications, which have different requirements in terms of network performance. The new architecture should provide different services and also the assurances that users require.

In this chapter we first review the main aspects of multiservice packet networks, including the concepts of Quality of Service (QoS) and service guarantees. Then we describe the set of network mechanisms that can be used for building network services. After that we present the main multiservice network architectures that have been proposed in the Internet, Integrated Services and Differentiated Services, and also the architecture of MultiProtocol Label Switching, which is a base technology to enable QoS. Then we deal with TCP elastic traffic, its characteristics and QoS requirements. This traffic is generated by the traditional "data" applications in the Internet, such as web browsing, peer-to-peer file sharing, ftp, e-mail and others. These applications are built on top of TCP, which provides reliable transfers and adjusts the sending rate to the network conditions to achieve the maximum possible throughput (it is because of this ability that TCP flows are called elastic). TCP elastic traffic requires the maximum possible throughput above a minimum value from the network.

2.1 An introduction to multiservice packet networks

In this section we provide a basic introduction to the main issues related to multiservice packet networks. After defining the concept of a multiservice network, we deal with the notion of QoS, which can be defined as a description of the performance, i.e., how well the functionality of an application is provided to its user, how well the local resources in hosts perform the different tasks of the application, or how well the packet flow is delivered by the network. Then we describe the differences in QoS requirements of "data" applications and real-time applications. After that, we explain the differences between services with absolute guarantees and services with no absolute guarantees, and we give some examples. Finally we

briefly describe two other important issues in multiservice networks: the role of Service Level Agreements (SLAs), which regulate the relation between a service user and a service provider, and the interconnection of multiple networks, each under a separate administrative control (multidomain context), to provide end-to-end services.

2.1.1 The concept of multiservice packet network

Telephony and file transfer are two examples of applications in which users have very different requirements for its performance. During a voice conversation moderately intermittent cuts and noise may be tolerated as long as intelligibility is not affected, but users are very sensitive to the lack of interactivity (a long time waiting for a response is very annoying). When using a "data" application like file transfer, moderately long transfer times may be tolerated, but errors are not. The result is that these two applications have different needs in terms of network performance, that is, the service provided by the network. Traditional networks were designed for specific applications, such as the telephone network for analog voice conversations, the TV network for video broadcasting and the Internet for "data" applications. These networks provide a very different service and have a very different architecture (the first and second one, circuit switching, and the third one, packet switching).

The arrival of digitization allowed audio and video to be stored and processed as classical "data" by computers, so that there are no fundamental differences between the transmission and manipulation of audio, video and "data". Instead of a specialized network, a single network could be built that is able to provide a set of different services suited to all kinds of applications. This is known as a multiservice network or an integrated service network, which has obvious advantages, such as large economies of scale. Packet switched networks obtain more efficient use of the network resources through statistical multiplexing, and therefore, there is a broad consensus that the goal is to build a multiservice packet network (or Integrated Service Packet Network, ISPN).

The topic of multiservice packet networks is not a new one in networking technology, as Asynchronous Transfer Mode (ATM) networks addressed it long ago. However, this topic has been introduced into the Internet only recently. Traditional usage of the Internet was dominated by "data" applications such as web browsing, ftp, e-mail or remote login. The success story of the Internet has given rise to the need for supporting all kinds of applications. New applications, with different requirements in terms of network service than the traditional ones, are foreseen, such as (Internet) telephony, video conferencing, (Internet) TV and radio, distance learning, collaborative working, shared virtual reality, electronic commerce, distributed games, and many others. The Internet is evolving to become a multiservice packet network.

2.1.2 Application QoS, system QoS and network QoS

Distributed applications are composed of a set of processes running in several end-systems (or hosts) connected through a network. Users attached to end-system exchange flows, that is, sequences of messages from one source end-system to one or more destination end-systems. These flows can be continuous flows (e.g., of audio or video, that are analog) or discrete flows (e.g., web requests and pages, user files...).

Multilayer architectures are normally used to describe distributed applications. The messages exchanged by users are processed by the application using local resources and turned into packet flows that are sent to and received from the network (Figure 2-1). The applications use the resources of the end-systems and the network that together constitute the "system". The

system manages all these resources and shares them between different applications running in the same and other hosts.

The concept of QoS can be defined as a description of the performance. The above multilayer architecture results in a QoS layered model, where the QoS is defined at different layers: the application QoS, the system QoS and the network QoS (Figure 2-1). Using layers and the interaction between applications and resources has long been studied [knoc97a, domi98a, nahr95a, voge95a, veci99a, schm97a, boch97a, aras94a, coul05aa].

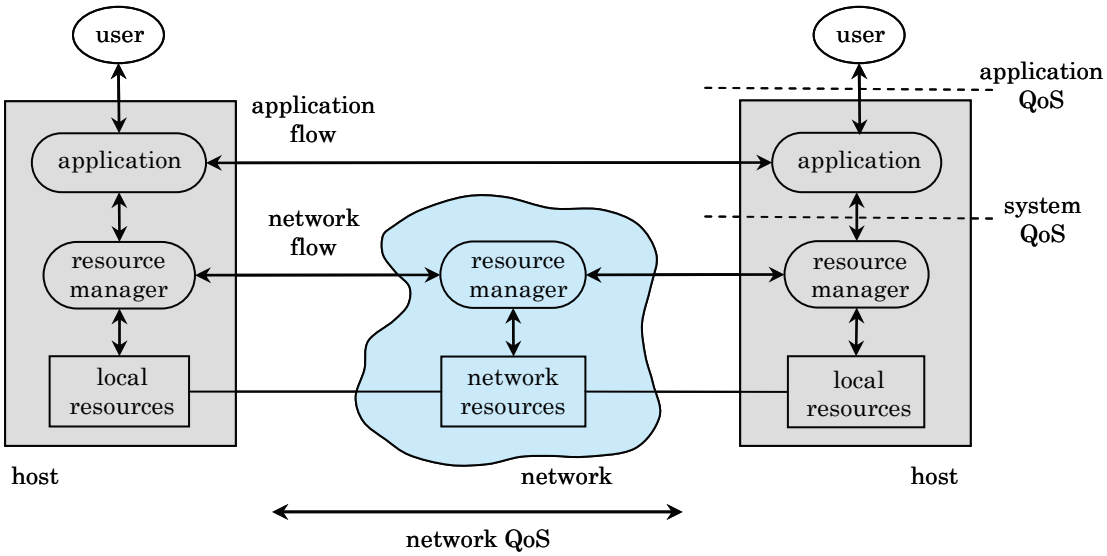


FIGURE 2-1: Application QoS, system QoS and network QoS.

Application QoS

The service of an application is the functionality provided to the user. The description of its performance, that is, how well this functionality is provided, is known as the QoS at the application layer. The performance can refer to:

- The fidelity to the original message, e.g., the voice intelligibility and its quality, the continuous playing of a video and its image quality, errors in a transferred file...
- The interactivity, e.g., the waiting times during a voice conversation or the response time between the request and the visualization of a web page...
- The time synchronization between flows, e.g., between several voices in audioconferences, the voice and movement of a pencil in a shared whiteboard, etc.

Application QoS is a measure of the satisfaction of the user, the difference between the user's perception and the user's expectations of the performance. Therefore, it is a qualitative measure that later is mapped into quantifiable parameters (e.g., no errors in a file transfer, the traditional telephone quality in telephony, a certain maximum response time in web browsing, etc.).

System QoS

Applications use the system to provide their service. They perform several tasks to process the messages (such as digitization, segmentation, coding, packetization, timing, error control, buffering, rate adaptation, presentation, etc.) and send and receive the flow of packets through the network. Applications use local resources for local processing and move packets to or from

network interfaces, and they use network resources to transport packets across the network. Examples of local resources are the processor, the memory, the file system, the bus, the screen and the communications hardware. Examples of network resources are the link's capacity and buffer, the processor and the memory in a router.

The service the system provides to the applications is the local processing and delivering packet flows. The functions of the system layer are to manage the local resources ("Operating System functions") and the network resources ("network layer functions"), in order to control sharing the resources between different packet flows.

QoS at the system layer defines the QoS the application gets from the system. The QoS parameters can be classified into two groups:

- In the first group the parameters measure how well the local resources perform the different tasks of the application, i.e., the processing time and memory assigned to a process, the screen properties (resolution, size, colors, etc.), and others.
- In the second group the parameters measure how well the packet flow is delivered, mainly relating to the packet delay and the packet loss (e.g., a maximum delay, a maximum percentage of loss, etc.). Below we deal in more detail with the QoS parameters of a flow, as well as the traffic parameters of a flow, which is a related point.

Network QoS

Packet delay and loss at the system layer come from two parts: one due to both source and destination hosts, and another due to the network. Hosts can introduce delays, bottlenecks, and the like, that are due to hardware or Operating Systems effects and have nothing to do with the network behavior. The concept of network QoS is defined in order to separate these two issues.

The service of a network is to provide a communications channel, that is, a path through which the packet flow will travel. The description of the network performance, that is, how well the flow is delivered by the network, is the network QoS. The performance is described by packet delay and packet loss as in system QoS, but instead of being defined at the host level, it is defined at network level, from the interface between the source host and network to the interface between the network and destination host. Like in the system layer the flow traffic description is also an important related point here at network level. Below we deal with the QoS and traffic parameters of a flow in more detail.

Traffic parameters and QoS parameters of a packet flow

The service provided to a packet flow, at the system and at the network layer, is its delivery. The definition of the service comprises two aspects: a description of the desired QoS and a description of the input traffic profile that receives this QoS, through a set of traffic and QoS parameters.

The traffic parameters of a flow try to characterize the temporal behavior of the flow. Each packet of the flow is defined by the following values (see Figure 2-2):

- The time instant of the first bit (or alternatively, the time between consecutive packets or packet inter-arrival time).
- The length of the packet.

From these values, several traffic parameters can be derived, such as:

- The average rate in a certain period of time (the aggregated length of packets divided by the period duration), the maximum or peak rate, etc.

– The average packet length, the average burst size, etc.

The QoS parameters of a flow aim to characterize how well the flow is delivered. For each packet, two main aspects are considered (see Figure 2-2):

– Packet delay, or the time from sending the packet (the time instant of the first bit) until it is received (the time instant of the last bit), and
– Packet loss, that is, when the packet is not delivered.

From these values, several QoS parameters can be derived, such as:

– The mean delay, the maximum delay, the minimum delay, percentiles, etc.
– Parameters about the delay jitter, i.e., about the delay variation, which can be defined in different ways: the difference between the delay of consecutive packets, the difference between the delay and the maximum delay (or a reference delay), and others. Then, parameters such as the mean jitter, the maximum jitter, etc., can be used.
– The percentage of loss, i.e., the ratio of the lost packets over the sent packets.
– The throughput, i.e., the average receiving rate in a certain period of time (this is a parameter related to the percentage of loss, because the ratio between the throughput and the average sending rate is equal to "1 – loss ratio").

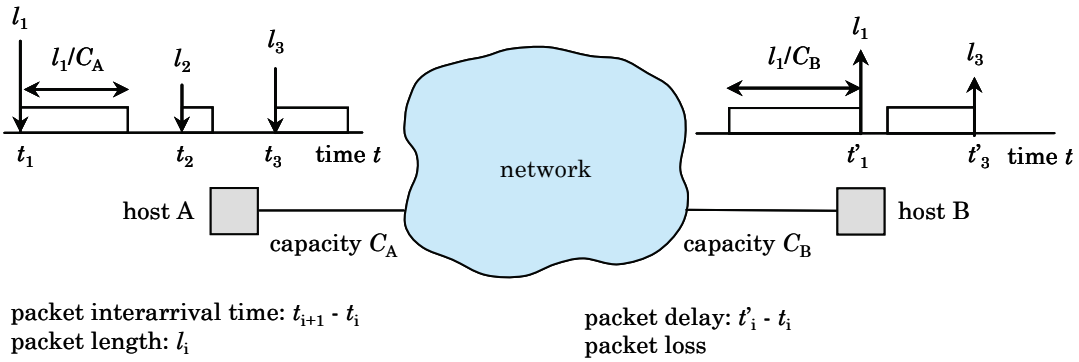


FIGURE 2-2: Traffic parameters and QoS parameters of a packet flow.

This set of parameters can be extended with parameters that measure the erroneous bits of delivered packets (e.g., bit error ratio), as well as others that measure aspects such as the reliability of the service or the service setup time. It is worth mentioning that in the Internet the IP Performance Metrics (IPPM) Working Group (within the Internet Engineering Task Force, IETF) is developing a set of standard metrics to provide quantitative measures of QoS, and also proposing measurement methodologies. The metrics considered by the Working Group are connectivity, one-way delay and loss, round-trip delay and loss, delay variation, loss patterns, packet reordering, bulk transport capacity, and link bandwidth capacity. A general framework is defined in [rfc2330] and more aspects in other RFCs (Request For Comments) available in [ippm].

The desired QoS in a service is described by the above QoS parameters and can be expressed in deterministic, statistical or qualitative terms [kuro93a, jami97b]:

– In deterministic (or "hard") terms, a set of QoS parameters has to be "always" met. For example, no packet has a delay greater than a maximum value, or no packet is lost, etc.
– In statistical (or "soft") terms, a set of QoS parameters has to be met during some specified percentage of time (during some defined time interval). For example, less than x % of

packets have a delay greater than a maximum value, or less than y % of packets are lost, etc.

– In qualitative terms, a set of generic QoS parameters without specifying target values has to be met in an unspecified percentage of time. For example, low loss, low delay, low jitter, etc.

2.1.3 The QoS of real-time and "data" applications

A classical and broad way of classifying applications is to differentiate between real-time (also called "streaming") applications and "data" (also called "elastic") applications:

– "Data" applications deliver discrete (time-independent) messages or "documents" such as text, images, files and other. Some examples are web browsing, peer-to-peer file sharing, ftp, e-mail, or telnet.

– Real-time applications deliver continuous (time-dependent) messages such as video and audio, which have to be played continuously. Some examples are TV, radio, telephony, videoconferencing, audio on demand or video on demand.

As we have seen in the previous subsection, application QoS is mainly described in terms of fidelity to the original message and in terms of interactivity:

– In "data" applications absolute fidelity is required, i.e., the users expect no errors in the transfer of documents. The interactivity here refers to the response time between typing a character at a telnet client and the visualization of the echo sent by the server, between the request of a web page and its visualization, or between an ftp file request command and the end of the file transfer. A telnet session requires high interactivity, web browsing requires medium interactivity, and ftp or e-mail requires low interactivity.

– In real-time applications there can be different levels of fidelity: in terms of intelligibility and quality in audio, or image quality and movement in video. For example, several audio quality levels are possible, such as the traditional telephone quality, GSM or CD, which take different coding formats and bit rates. In video, different coding formats such as the MPEG family, the number of visualized frames per second, the image size, and the color resolution, are parameters that determine different fidelity levels. Moreover, errors and losses in audio and video reduce the fidelity perceived by users, although there can be a certain degree of tolerance (e.g., cuts in telephony may be tolerated as long as the intelligibility is not seriously affected). The interactivity here refers to the waiting times in telephony, the response time to commands such as pause, rewind, etc., in audio or video on demand, or other. Telephony needs high interactivity, since otherwise the conversation is not possible. Audio and video on-demand needs low interactivity, as well as audio and video broadcasting.

The applications process the messages exchanged by users with several mechanisms and then turn them into packet flows that are sent to the network (and the reverse procedure). The following are some of the mechanisms the applications may use:

– In "data" applications, partitioning of documents in blocks, packetization, sequencing, error control using retransmission to achieve absolute fidelity, rate adaptation at the sender to achieve the maximum possible network throughput (and reduce the response time), presentation to the user, etc.

– In real-time applications, (audio and video) digitization, coding, compression, segmentation, packetization, timing, sequencing, error control using retransmission or using receiver based techniques, buffering at the receiver to reduce the delay jitter, delay adaptation at the receiver to balance fidelity and interactivity, rate adaptation at the sender to match the actual network throughput, encoded layering, presentation to the user (e.g., playing continuously audio and video), etc.

The flow of packets is sent to and received from the network. The required network QoS (the same holds for the system QoS relative to the flow delivery) is the following:

- "Data" applications deal with packet loss through error control techniques based on retransmission, while packet delays are not so important. The network throughput is the most important QoS parameter since it mainly determines the response time (the interactivity). Moreover, sources increase and decrease the sending rate according to the network conditions to achieve the maximum possible throughput (this is why they are called "elastic" flows). Therefore "data" applications require a minimum network throughput and can benefit from any increase of throughput.
- Real-time applications send flows with intrinsic characteristics that the network should preserve (these are called streaming flows). They require a maximum jitter and a maximum percentage of loss, and depending on the interactivity, a maximum delay (the required assurance depends on whether they are delay-adaptive or not). Sources can send at a constant rate or at a variable rate (through compression techniques, detection of silence, etc.), and some can also adjust the sending rate (within a range) to the network conditions.

Finally it is worth commenting on the following two points about the relationship between the QoS at different layers:

- The application QoS required by the user depends on the nature of the application and also on the user's desires.
- The system (and network) QoS required by the application depends on the required application QoS and also on the implementation of the application.

The required application QoS is a qualitative measure that is mapped to quantifiable parameters. For example, a "good" QoS for a user in videoconferencing can translate to a minimum number of visualized images per second, some image quality parameters (resolution, size, number of colors...), and a maximum response time. The point is that this mapping depends on the application but also on the user's desires. A file transfer requires extreme fidelity that it is not so necessary in a videoconference, and the degree of interactivity in a voice conversation is higher than in web browsing. A demanding user could want more interactivity when browsing (e.g., in a company) or better voice quality than other users.

The second point deals with the topic of mapping application QoS parameters and system (and network) QoS parameters. In the above example, videoconferencing requires sending a number of packets per second to be received with a maximum percentage of packet loss and a maximum packet delay. The translation depends on how the application is made. For example, when in an audio application a voice packet is lost (e.g., discarded in the network), one application could play nothing (silence), another could repeat the previous packet, or another one could interpolate from previous and future packets. In each case the offered application QoS is different; therefore, some of these applications can tolerate a higher degree of packet loss. Some real-time applications implement delay-adaptation techniques at the receiver to achieve a desired balance between late (lost) packets and interactivity. "Data" applications and some real-time applications implement rate-adaptive mechanisms that reduce the sending rate to keep delays and losses moderate. Therefore, some applications can gain more than others from the same system and network QoS (or in another way: a better implemented application can compensate a worse system and network QoS). Obviously, the application alone cannot do the entire job and a minimum system and network QoS is always required; however, studying its possibilities is very interesting in order to define network services. Of course the better the system and network QoS is, the better the application performance is.

2.1.4 Service guarantees

As we have seen in Subsection 2.1.2, the definition of a network service comprises two aspects: a description of the desired QoS and a description of the input traffic profile that receives this QoS, through a set of traffic and QoS parameters, in deterministic, statistical or qualitative terms. In this subsection we deal with another important aspect in the definition of services: the kind of guarantee of the service. We distinguish between services with absolute guarantees and services with no absolute guarantees. Although we focus on network services, these concepts could be easily extended to other QoS layers.

Service with absolute guarantees

A service with absolute guarantees means that the desired QoS is assured during the flow's lifetime, i.e., it is not possible that the flow receives the desired QoS for a certain time and later the provided QoS gets worse. Therefore, an application can count on a specific performance of the network.

These are some examples of network services with absolute guarantees:

- A guaranteed delay service, where (in-profile) packets have an absolute assurance on delay, in terms of the maximum delay, jitter, or other, and expressed in deterministic, statistical or qualitative terms. Usually no loss or a small maximum percentage of loss is guaranteed. Out-profile packets do not have the same guarantees, and depending on the service, they can be delivered with undefined delay, not delivered or even definitely discarded. For example, a network service that guarantees a packet delay that is always smaller than 150 ms (deterministic terms) for input traffic that never exceeds 250 Kbps of peak rate.
- A guaranteed throughput service, where (in-profile) packets have an absolute assurance on throughput (on loss). There is no guarantee on delay, which usually can be high. Out-profile packets do not have the same assurance on delivery, and depending on the service, they can be delivered if it is possible or even definitely discarded. For example, a network service that guarantees a throughput of about 200 Kbps, and provides an extra throughput if possible, i.e., if the average sending rate does not exceed the guaranteed throughput there will be almost no packet loss (qualitative terms), and if it does exceed it, the exceeding traffic can be delivered.

Service with no absolute guarantees

A service with no absolute guarantees means that the desired QoS is not assured during the flow's lifetime, i.e., it is possible that the flow receives the desired QoS during some time and later the provided QoS gets worse. Therefore, an application cannot count on a specific network performance, although the provided QoS may or may not meet the requirements of the application.

The following are some examples of network services with no absolute guarantees:

- A relative service, where the QoS received by a flow is defined as a function of the QoS received by other flows (e.g., the mean delay provided to a flow is, at least, half the one provided to another flow; or the throughput is, at least, twice). It is said that there is a "relative guarantee". In a relative throughput service the flow has an assigned weight, so that throughputs and weights are proportional (double weight, double throughput). Similarly, in a relative delay service, delays and weights are proportional.
- A best-effort service, which attempts to achieve the best service, but there is neither absolute QoS guarantees nor "relative" ones, and the provided QoS can be whatever.

2.1.5 Service Level Agreements (SLAs)

The relation between users and providers of network services must be regulated. A Service Level Agreement (SLA) is a contract between a user and a provider that defines a large quantity of aspects of the service delivery [rfc2475, alip05a, prof01b, rfc2638, trim01a]. This includes a technical part (also called Service Level Specification, SLS), in terms of the characteristics and availability of the services the user can ask for, and a non-technical part, in terms of pricing, penalties or verifying methods.

For the technical aspects, the user manages service requests according to a policy and the provider checks that user's requests are within the contract. User's expectations about the service are specified in the agreement. To meet these expectations, the provider plans the provision of resources for the services taking into account the service agreements and user's behavior. Moreover, methods to verify the agreement need to be defined, i.e., some criteria for an accurate common understanding between users and providers for the performance and reliability of the services they use or provide. Therefore, a set of standard metrics must be specified to measure these quantities. Measurements are used to track the provided services, and also as an information source about service usage for problem diagnosis and resource planning.

The following are some of the aspects that may be included in SLAs (technical and non-technical):

- Services that can be requested.
- Traffic profile parameters for each service, i.e., the traffic parameters (average sending rate, maximum burst size, peak rate, maximum packet length, etc.) that define the in-profile traffic. This can be specified in the long or sort-term (through signaling).
- QoS parameters for in-profile traffic (throughput, percentage of loss, maximum delay, jitter, the setup delay, etc.), and also, the QoS for out-of profile traffic. This can be specified in the long or short term (through signaling).
- Duration of the service, such as permanent, on demand, or scheduled in advance.
- Scope of the service, that is, a set of locations where traffic can be sent to and a set of locations where traffic can be received. Destination/s for output traffic and source/s for input traffic can be predetermined or not (e.g., anywhere in the Internet, or a virtual leased line between two specified points).
- Traffic profile, QoS, duration and scope of the service can be specified on a long-term basis or on a short-term basis through signaling (on demand).
- Service availability, e.g., the period of time during which the service may be used.
- Service accessibility, e.g., the probability of admission refusal.
- Reliability, mean time to repair.
- Other aspects such as encryption, authentication, routing constraints, etc.
- Mechanisms for monitoring and auditing the service.
- Fees, pricing and billing mechanisms.
- Responsibilities of the two parties, remuneration if these responsibilities are not met, and arbitration policies.
- The frequency of report generation and the level of reporting detail.
- The parties involved in the agreement (the SLA is a legal document).
- The terms of the agreement and modification terms.

2.1.6 Interconnection in a multidomain network

Networks are made of the interconnection of multiple networks, each one under a separate administrative control, called domains (or Autonomous Systems, AS, in the Internet). All aspects of multiservice networks, service building, QoS policy, charging, agreements, planning, and so on must be considered in this multidomain context.

Inside the domain each administrative control decides how to provide the services, the QoS policy, protocols, or any other aspect. Neighboring domains need to cooperate to provide an end-to-end service when the end-to-end path crosses boundaries between domains. Each domain relies on its neighboring domain to deliver its traffic, and in turn this neighboring domain passes the traffic to its neighbors, etc., until the destination is reached. The interconnection of domains is based on SLAs. The previous subsection dealt with SLAs; however, for a multidomain context a generalization is needed. Interconnection models between providers are discussed in [geor04a].

A (primary) provider provides a service to an end-user, but to do this, this provider has to rely on other (sub)providers so that it acts as a user of these other providers. Therefore, the term user can refer to an end-user, a network or a group of networks. Multilateral agreements rarely work, so to scale the administrative complexity of inter-domain QoS, end-to-end services are built from concatenations of bilateral agreements [geor04a, zhan98c, jaco98a, eure99a]. Bilateral agreements are based on the "one-stop responsibility". For the end-user, the primary provider handles the service it provides globally, and the service is only ruled by their agreement. Then the primary provider, acting as a user, has another user-provider agreement with each of these (sub)providers. The "one-stop responsibility" can be recursively applied to these agreements. This allows end-to-end services to be provided in a simple way while stating clearly the responsibilities if there are agreement violations. Therefore, the definition of SLAs with neighboring domains in a multidomain context plays a central role in providing end-to-end services.

2.2 An overview of the mechanisms for building network services

The Internet architecture requires more complex network mechanisms in order to become a multiservice network. In this section we describe the mechanisms for building network services in packet switching networks. Firstly the causes of packet delay and loss in statistical multiplexing are explained. After that we present the set of network mechanisms that can be used for service building, that is, queue disciplines, traffic conditioning, admission control, routing, management and provisioning. Then we discuss the relation between these network mechanisms and the service guarantees. Finally we deal with the concepts of per-flow and per-aggregate state, and the differences between core-stateful and core-stateless networks.

2.2.1 Delay and loss in packet switching networks

Packet switching or "store-and-forward" networks use statistical multiplexing in the links. Sources send sequences of packets and switches maintain a queue for each output link. Upon receiving a packet, the switch puts it in the corresponding output queue, together with other packets. Finally, packets are taken from the queue and sent at the rate of the link's capacity.

Statistical multiplexing takes advantage of the bursty nature of the traffic to achieve better efficiency. Capacity can be allocated to flows in terms of the average rate (or similar) instead of the peak rate, because queues can absorb traffic bursts. However, queuing may cause variable delay and even losses; thus, affecting the QoS the flow receives. During some periods of time the arrival of packets may be greater than the link's capacity, so packets are queued, and

therefore, delayed. If there is not enough space in the queue, some packets are discarded. During other periods of time it is expected that the arrival of packets will be smaller than the link's capacity, so queued packets will be sent. This causes variable delays and losses.

To sum up, the delay and loss experienced by a packet passing through a packet switching network are due to the following:

– The delay can be split into several parts. One is the propagation time of the electromagnetic signals through the physical medium, which depends on the light speed and the traveled length, in all links. The second part is the transmission time at each switch waiting for the entire packet to arrive, which depends on the link's capacity and the packet length. The third part is the processing time of the packet in each switch, mainly to perform a lookup operation in a table to know how and where to forward the packet. The fourth part is the queuing time, or the time the packet waits in the output queues of switches until it is transmitted. Unlike the others, queuing time is variable and considered difficult to predict and control.
– A packet is lost when it is discarded in a switch, mainly because there is no free space in queues, but also because errors are detected in the packet caused by unreliable links (especially wireless ones).

2.2.2 Network mechanisms for service building

A network service definition comprises a description of the desired QoS and a description of the input traffic profile that receives this QoS, through a set of traffic and QoS parameters, in deterministic, statistical or qualitative terms. Queuing delays and losses experienced by a flow passing through a network path, depend on the resources of the path (capacity of links and buffer length of queues) and on the interactions of the flow with other flows encountered on the path (see Figure 2-3). All these aspects depend on a set of network mechanisms that are therefore used for building network services. These network mechanisms are the following:

– Provisioning, which determines network resources on long time scales.
– Management, which allocates and configures network resources on middle or short time scales.
– Routing, which decides the path the flow will follow; it is applied at flow time scale, and also to higher time scales.
– Admission control, which evaluates if the network can provide the requested service to the flow while maintaining the service promised to the other flows; it operates at flow time scale, and also to higher time scales.
– Traffic conditioning, which enforces that the traffic of the flow entering the network follows the traffic profile agreed for the service; it operates at packet time scale.
– Queue discipline, which decides when a packet is sent and when a packet is discarded, if necessary; it operates at packet time scale.

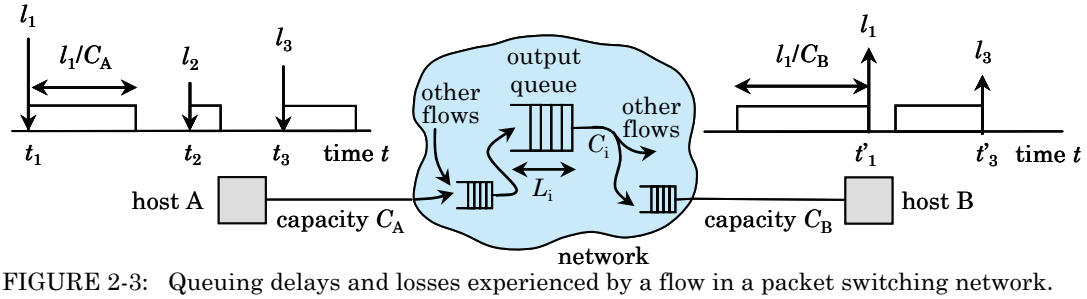


FIGURE 2-3: Queuing delays and losses experienced by a flow in a packet switching network.

The goal of these mechanisms is to provide the required network services, to achieve high resource utilization and to minimize the cost.

Network provisioning

Provisioning determines the resources of the network at long time scales, while management allocates and configures these resources at middle or short time scales. The goal of provisioning is to ensure that the network has enough resources to meet the expected demand with adequate QoS. Users expect some accessibility to the service, usually high, and providers need to plan and update the network to meet these expectances. Updating the physical network topology is based on performance measures, detection of physical bottlenecks, potentially dangerous nodes and links in the case of failure, and others. Pricing is fundamental for giving incentive to the provider to expand capacity as demand grows in order to ensure that the performance is maintained. Therefore, it is very important to understand the relationship between the user's demand, resources and performance.

Network management

Network management deals with maintaining correct operations in network infrastructures. It operates at middle or short time scales. A typical description of network management functions would include the configuration and control of the equipment and software, with a close relationship with the other network mechanisms (queue disciplines, conditioning, admission control and routing), from a higher level. It also includes mechanisms for maintaining a high network performance, i.e., trying to carry as much traffic as possible using the same network resources without service degradation. Another management function includes mechanisms for detecting network failures and restoring them as fast as possible when they occur. Accounting for network utilization is also an important management function for several reasons, such as detecting problems, charging users, etc.

Routing

This decides the path the flow (or the aggregation of flows) will follow between the different possible paths. In QoS routing, the returned path is the one that is most likely to meet some QoS requirements [chen98a, xiao00a, xiao99a, rfc2386]. Constraint-based routing extends QoS routing by considering other constraints of the network such as policy or the goal of increasing the network utilization. Since it considers other criteria in computing routes than the minimum path in terms of hops, it may find a longer but more lightly loaded path that is better than a heavily loaded shorter path. It considers not only topology, but also the requirements of the flow (or flows), resource availability of the links, and other possible policies specified by network administrators.

A router needs topology and resource availability information such as available capacity in order to compute QoS routes. A routing protocol distributes this information. Since resource availability changes frequently in time, the frequency of messages should be reduced to the situations when there are topology modifications or significant changes in resource usage. Moreover, computing new routes depends on the metrics chosen, e.g., hop count, available capacity, delay, jitter, or others. Using multiple constraints can cause a big routing table and a high computation overhead that can make the routing problem very complex. Reducing the number of constraints and the computation frequency can reduce the complexity of routing computation.

Admission control

A new flow (or an aggregation of flows) arriving to the network edge wants to receive a given network service (with absolute guarantees). The requested service comprises a description of the QoS and a description of the input traffic profile that receives this QoS. Network routing finds a path through the network towards the destination. AC evaluates if in the chosen path it is possible to provide the requested service to the new flow while maintaining the service already promised to the actual flows in the path. The AC decision must take into account the QoS requirements and traffic characteristics of the new flow and the QoS requirements and traffic characteristics of the actual flows in the path. If the flow is accepted, it will receive the desired service during its lifetime. If it is rejected, the service is not provided, although renegotiations about the traffic profile or about the QoS (degrading it to a lower one, or even to provide the best-effort service) are possible. Note that in the case of acceptance and in order to isolate (or protect) the rest of the flows, the network should check the flow's input traffic and enforce the agreed traffic profile (e.g., discarding non-compliant packets), so that if the flow exceeds the traffic profile, it does not damage well-behaved flows. This can be achieved through traffic conditioning mechanisms and/or some specific queue disciplines in the network. We deal with AC in more detail in Chapter 3.

Traffic conditioning

For a given a service, there is an input traffic profile description that allows each packet to be classified as an in-profile or out-profile packet. In-profile packets receive QoS while out-profile receive another (lower) QoS depending on the service's rules. Therefore, traffic conditioning involves some traffic metering and comparison to a profile, and some actions:

- The traffic profile can be based on traffic parameters such as the peak rate or the average rate. Traffic may be measured using algorithms such as the token bucket [clar92a], the Time-Sliding Window (TSW) [clar98a], and others.
- The actions can be delaying an out-profile packet until it becomes an in-profile packet (also called shaping), discarding an out-profile packet (also called policing), or marking a packet, i.e., setting a packet field to a specific value, depending on whether the packet is in-profile or out-profile and on the agreed QoS. Packets with the same mark are called a "packet class".

Traffic conditioning can be applied to a single flow or to an aggregate of flows. It can only be applied at the edge of the network or at each node of the path (note that multiplexing changes the traffic profile, so an in-profile packet at the network edge can even become an out-profile packet in the core).

Queue disciplines

Flows that meet together in the same link share its capacity and buffer according to a certain queue discipline. The discipline is a set of rules that can be divided into two parts:

- The scheduling algorithm, which decides when a packet is sent.
- The queue management algorithm, which decides when a packet is discarded if necessary.

The queue discipline can apply a different treatment to different "groups" or sequences of packets, which corresponds, for example, to a flow, an aggregate of flows, a packet class or other. Therefore, incoming packets need to be classified appropriately before applying the different treatment.

There is a lot of literature on queue disciplines and research is still going on. Some examples of schedulers are the following [stil96a, zhan95a, bhat00a, lieb99a, zhan91a, stoi99a, deme89a, pare93a]:

- First-In-First-Out (FIFO). Packets are transmitted in the same order in which they arrive at the queue.
- Priority. Packets are classified into one or more priority "groups", and each one has its own queue. The packet that is transmitted is chosen from the highest priority queue that is non-empty.
- Fair Queuing (FQ) and Weighted Fair Queuing (WFQ). Packets are classified into several "groups", and each one has its own queue. In FQ the discipline alternates service between the queues, so that the "active" ones (that is, the queues that have packets) share the link equally ("fairly"). In WFQ the scheduler also serves ("active") queues in a circular manner, but the difference is that the service is not shared equally, since each queue receives a portion of the link proportionally to a given queue weight (in a "weighted fair" way). FQ and WFQ provide isolation (protection) between the different "groups".
- Virtual Clock, Delay Earliest-Due-Date, and others.

Some examples of queue management algorithms are the following [labr99a, bhat00a, stoi03a, floy93a, clar98a]:

- Tail Drop. This simply drops the last packet when there is no available buffer space.
- Active Queue Management mechanisms, such as Random Early Detection (RED). RED starts to probabilistically drop packets before the queue is full, by using a dropping probability that depends on the average buffer length. An extension is RED with In and Out bits (RIO), where packets are classified into two priority "groups" (actually, two classes), and one priority "group" is preferentially (and probabilistically) discarded over the other.
- Core-Stateless Fair Queuing (CSFQ), and others.

2.2.3 Service guarantees and network mechanisms

The traditional network scheme in the Internet provides the best-effort service. The scheme is very simple. There is no control of the traffic sent by sources, neither traffic conditioning nor AC. The notion of flow does not exist. The queue discipline is FIFO and Tail Drop, which do not differentiate between different flows or "packet classes" (one could say that there is "one packet class"). Each packet is sent as soon as possible, so it tries to do the best. Occasionally, the traffic load can grow and exceed the resources, output queues get full (this depends on the ratio between the link's capacity and the arrival rate), and delay and losses may increase to a level that does not meet the QoS requirements of applications. Therefore, absolute guarantees are not provided. Moreover, FIFO alone does not provide any isolation between flows, and if one flow injects excessive traffic, the service for the rest is completely disrupted. The throughput a flow obtains is proportional to its sending rate; therefore, it encourages misbehaving users.

This simple scheme can be combined with rate-adaptive mechanisms in the applications to share network resources fairly between all flows. This is the case of "data" applications in the Internet that use TCP. TCP decreases the sending rate in the case of packet loss and increases it otherwise in order to achieve the maximum possible throughput while fairly sharing resources [jaco88a]. The goal is to provide a fair throughput service (a relative service), which provides a throughput equal to the fair rate in the bottleneck link, i.e., the link's capacity divided by the number of present flows (the effective TCP resource sharing may, however, exhibit unfairness in some situations). Therefore, as the number of flows increases, the provided throughput decreases (and the time to transfer a document increases); this may be

below an acceptable minimum throughput. Some real-time applications can also use rate-adaptive mechanisms, but again there is also a minimum acceptable quality that imposes a limit on reducing the sending rate. The consequence is that occasionally the QoS provided may be below the minimum requirements of flows, and therefore services with absolute guarantees are not provided. Moreover, relying on end-systems without any network control has some drawbacks since isolation between flows is not provided. Not all sources can be implemented to react when the QoS gets worse, or to react in the same way. Therefore, some flows can achieve a better QoS at the expense of the rest and the network does not control this.

The next step is changing FIFO and Tail Drop by other queue disciplines in order to introduce differentiation between different "groups" or sequences of packets, e.g., flows, aggregates of flows, packet classes (by using marks in the packets' headers previously written at the network ingress) or others. Routers classify packets into these "groups" and each one receives a different treatment in the output queues. For example, using two priorities in the scheduling, high priority packets always get a smaller delay than low priority packets (e.g., to favor real-time flows). However, "smaller delay" for the high priority packets may mean 2 s or 20 ms, depending on the actual high priority traffic and the existing resources. Another example would be using a FQ scheduler, so that the different "groups" get the fair share of the link's capacity (using WFQ, a weighted fair share). However, the final share can be anything depending on the number of groups and the existing resources. Another example would be using two priorities in the discarding, so that low priority packets are discarded first if buffers are full. As before, the final losses can be anything depending on the high priority traffic load and the existing resources. In all these cases, we say that the scheme provides a relative service, where the QoS received by a flow is defined as a function of the QoS received by other flows (see subsection 2.1.4). In conclusion, with the introduction of other queue disciplines, these schemes are able to provide different (relative) services and still remain simple in many cases. Furthermore, as in the best-effort case, occasionally the provided QoS may be below the minimum requirements of flows, and therefore services with absolute guarantees are not provided.

In the above schemes for best-effort and relative services, the ratio between traffic load and the existing resources is not controlled and can be anything. This is why the provided QoS may occasionally degrade and services with absolute guarantees are not provided. Sometimes the traffic load is not high and the resources are enough to satisfy the flow requirements during the flow's lifetime. In other occasions there is a traffic overload, and the resources are not enough to satisfy all flows. We say there is congestion. In the above schemes the resources during congestion are shared between all flows and none of them is satisfied. Congestion situations can happen more or fewer times depending on the behavior of users' demands, the chosen network provisioning and the routing techniques used. This can imply the violation of the user-provider agreement, depending on the tolerance to unsatisfied requests specified in the agreement (user's expectations). Congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques.

Congestion situations can be avoided by using over-provisioning, i.e., provisioning the network resources so that congestion never or rarely occurs. Using over-provisioning, any of the above simple schemes provides services with absolute guarantees, i.e., the QoS is assured during the flow's lifetime so that it is not possible that a flow receives the desired QoS then later the provided QoS gets worse (see subsection 2.1.4). Also note that besides providing absolute guarantees, all service requests are always satisfied. Over-provisioning could be achieved in the following way. Suppose that there is an agreement about the maximum traffic users may send (to anywhere), and that traffic conditioning mechanisms at the edge enforce these aggregated traffic profiles for the users' traffic. Since traffic flows are dynamic, a source can generate traffic to different destinations at different rates and the paths followed to the destinations may also change. Therefore, it is necessary to provision the network for the worst-

case scenario, e.g., when the maximum traffic is sent from all ingress points to the same egress point, etc.

If instead of over-provisioning, a "normal" provisioning is used, another option is dealing congestion situations with AC. In the above schemes for best-effort and relative services, AC is not used, and an overload situation is dealt with the "sharing" way, that is, the "few" resources are allocated to all flows, and therefore none of the flows receives the desired QoS. The "blocking" way is employed when using AC: resources are allocated to some of the flows (they are "accepted"), which receive the desired QoS, while the rest of the flows are not satisfied (they are "rejected" or "blocked"). According to [shen95a], if a minimum QoS is required, "blocking" is better than "sharing" when congestion occurs, since more flows are satisfied. Moreover, the accepted flows receive a service with absolute guarantees. In conclusion, using a "normal" provisioning, when congestion occurs, AC achieves efficient use of network resources by maximizing the number of satisfied flows and provides absolutes guarantees. However, using AC complicates the network scheme.

Over-provisioning the network resources and using AC are two opposed options (both allow services to be provided with absolute guarantees). The question about which option is better in the Internet has long been discussed, below we summarize the main arguments [bres98a, shen95a, bake97a, rfc1633, bona02b]:

- Adding AC complicates the network scheme, while over-provisioning does not require any modification and simple network schemes can be used.
- By using over-provisioning all service requests are satisfied, while using AC a percentage is blocked. The blocking rate depends on the behavior of users' demands, the chosen network provisioning and the routing techniques used (and also on the ability of the AC mechanism to maximize the number of satisfied flows). It therefore has to be carefully chosen.
- Over-provisioning can be very inefficient in using resources (and therefore more costly), while using AC with "normal" provisioning may achieve higher levels of utilization (that are less costly). Over-provisioning is based on a worst-case situation, and therefore it can be highly wasteful if this situation is very unlikely to happen. That is, if the variations in traffic demand are large compared to the average demand, over-provisioning requires a large increase in resources, and as a consequence the average utilization is low. The non-blocking availability of the services would result in more costly services for the users.
- There are several difficulties in achieving over-provisioning since some unexpected events that may not have been taken into account in a "nominally" over-provisioned network can happen. This includes inaccurate traffic forecasts, the sudden deployment of a new application or a change in application usage, routing changes, or most importantly link or router failures.

2.2.4 Core-stateful networks versus core-stateless networks. Class-based networks.

Network state is a general name that refers to the information about the flows, their service and their use of resources, used by the network mechanisms to provide services. This includes information for identifying a flow, service requirements, traffic conditioning parameters, parameters of the queue discipline, path to follow, utilization of link's capacity and buffers, etc.

The state is said to be per-flow based or per-aggregate based. Per-flow state means that the information is associated with an individual flow. For example, the value of the peak-rate of a flow used for traffic conditioning at the edge of the network, or some queue parameter that defines the share of the link's capacity for a flow. Per-aggregate state means that it is associated with an aggregate of flows, for example, the next router for a set of flows with destinations in the same network, or some queue parameter for a set of flows that receive the same treatment in queues. Both possibilities may coexist in the network in different places.

The state is distributed in the different elements of the network (edge and core routers). It can be configured statically or dynamically. Configured statically means that it is permanent, or at least in the long-term ("implicitly"). The network administrator configures it through previous signaling or manually. An example is the two end points of a virtual leased line, a same minimum throughput desired for each of the web flows sent by a user (identified as an aggregate with source address and port, and under the same traffic conditioning rules), or a queue parameter that defines the treatment received by a packet class. Configured dynamically means that it is done on demand, in the sort-term. It is configured through signaling ("explicitly") or through the same data packets ("implicitly). An example is the source and the destination of a usual phone call, a web flow, etc., a different minimum throughput desired for one ftp flow than for another flow, or a queue parameter needed to treat a flow individually. Both possibilities may coexist in the network in different places.

An operation that is closely related to the network state is packet classification (also called filtering). Given some rules that define a sequence of packets (a flow, an aggregation of flows, a packet class, etc.), packet classification consists in finding the sequence of packets a packet belongs to. Network elements perform classification, read the related state, and then operate accordingly (traffic conditioning, queue discipline, AC, etc.).

Packets are classified by inspecting one or more packet header fields and carrying out a matching operation based on the rules that define the sequence of packets. This matching operation can be simple or complex. For example, the classification in virtual circuit networks uses one header field and is simple, while the classification in IP datagram networks also uses one header field but it can be more complex (because the state is aggregated to reduce the routing tables using IP network prefixes). If the queue discipline applies a different treatment to each individual flow, a per-flow state is needed. Having a per-flow state based on the 5-tuple in IP (source and destination address, protocol, source and destination port) implies a complex classification. If the queue discipline applies a different treatment not to a single flow but to a "packet class", a per-aggregate (per-class) state is possible. With a small number of different treatments, the per-aggregate state, which is based on just one header field, implies a simple classification. Both possibilities may coexist in the network in different places.

Finally, the explanation above lets us explain the difference between a core-stateful network and a core-stateless network, and its implications:

- Core-stateful networks have a per-flow state in the core. Core routers (and also edge routers) use per-flow states to perform traffic control operations over all passing flows (see also subsection 2.3.2). A per-flow state in the core allows finer control, and therefore possibly better services and better resource utilization. However, some operations can become more difficult and exhibit low scalability with the number of flows: the processing of per-flow signaling is needed to setup the state, and also the above mentioned complex per-flow classification is needed. Therefore, these networks can be unfeasible if the number of flows to handle is very large.
- Core-stateless networks do not have per-flow state but per-aggregate state in the core. Edge routers may maintain per-flow state and assign each flow to one of a small set of aggregates. Core routers only differentiate between aggregates. An example is the networks based on packet classes, where flows' packets are assigned to a small number of classes at the ingress (a mark that identifies the class is written in the packet's header), and queue disciplines in the core apply a different treatment to packets belonging to different classes (see also Subsection 2.3.3). The aggregate control in the core cannot provide the same good services as in core-stateful networks, but they are simple and highly scalable since per-flow processing is not used and only a simple classification is necessary.

2.3 Multiservice network architectures in the Internet

The IETF is guiding the evolution of the Internet to become a multiservice network. The efforts are in two areas: modifying the network architecture to provide a better service model, and developing new protocols to support real-time applications (e.g., the Real-time Transport Protocol, RTP, and its companion, the RTP Control Protocol, RTCP [rfc1889]). In this section we will focus on the first of these research areas: building a new service model in the Internet. Firstly we review the architecture of the "traditional" Internet. Then we describe the main multiservice network architectures that have been proposed in the Internet, Integrated Services (Intserv) and Differentiated Services (Diffserv), which fundamentally differ in using the per-flow state in the network core. Finally we describe MultiProtocol Label Switching (MPLS), a network architecture based on using logical paths, which is not a multiservice architecture in the same sense as the previous two, but rather a base technology for giving support to mechanisms for optimizing network resource utilization and performance, for controlling the network QoS and for increasing the network reliability.

2.3.1 The "traditional" Internet

The Internet is an internetwork, which is a set of networks (or subnets) of different technologies that are connected by routers, and which is based on the TCP/IP architecture. In this architecture, routers and hosts share a common internetworking layer that interacts with the different subnet layers to create the "links" router-router and router-host (Figure 2-4). TCP/IP philosophy is to make the network very simple and robust, and put the complexity in the end-systems [salt84a, reed00a, blum01a]. Functions that are more complex such as error control, flow control, etc., are performed by layers above the internetworking layer that are only present in the hosts.

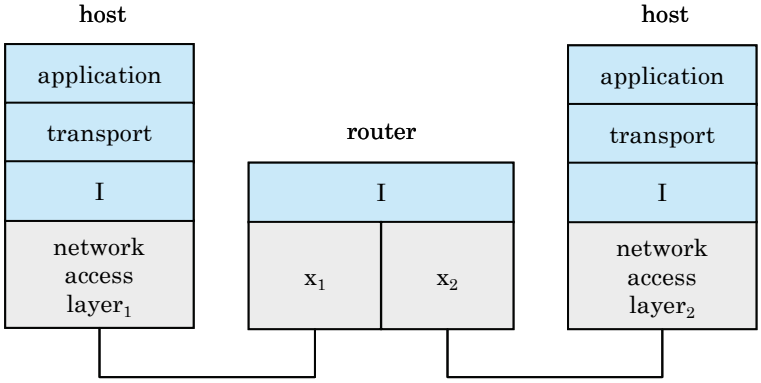
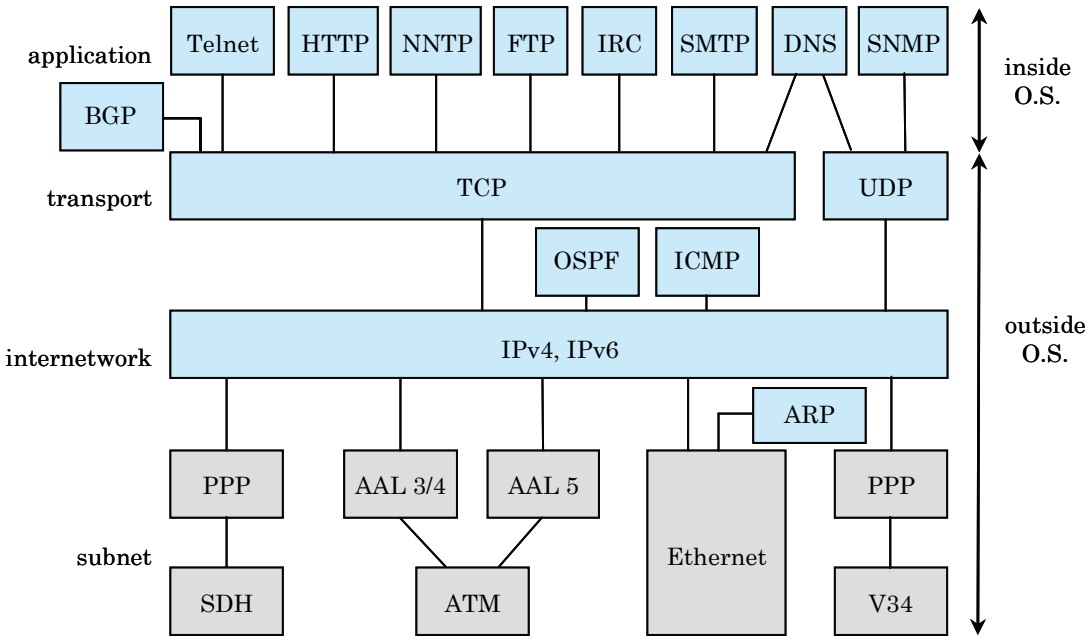


FIGURE 2-4: The layers of the TCP/IP architecture.

The internetworking layer, which is based on IP (Internet Protocol), provides a very simple service and can work with any type of subnet because it requires very little from them. This layer provides a connectionless, unordered, unreliable and best-effort service. Packets may be lost, duplicated, delayed or delivered out-of-order, but the service will not detect these conditions, nor will it inform the source or the destination. There is no control of the traffic sent by sources. Routers treat all packets in the same way without recognizing any differences between them. Each packet is sent as soon as possible, so it tries to do the best. FIFO and Tail Drop are used in queues. Moreover, the traditional network layer in the Internet uses the datagram mechanism and simple routing (e.g., based on the shortest path in terms of hops), which is not flexible enough to manage the network efficiently.

The transport layer includes two protocols (Figure 2-5), TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). Both protocols multiplex traffic from different application's processes through the ports. TCP provides a connection-oriented, reliable and ordered service, while UDP provides a connectionless, unordered and unreliable service. TCP reliability is achieved through retransmission techniques. Moreover, TCP uses rate-adaptive algorithms to achieve the maximum possible throughput while sharing network resources fairly between TCP flows [jaco88a]. The combination of best-effort and rate-adaptive algorithms aims to build a fair throughput service, which provides a throughput equal to the fair rate in the bottleneck link, i.e., the link's capacity divided by the number of flows present (however, the effective TCP resource sharing may exhibit unfairness in some situations). Therefore, if the number of flows increases, the provided throughput may decrease without any limit.

Traditional "data" applications like ftp, web, etc., are TCP-based since they need a reliable service. Real-time applications do not perform well with TCP due to excessive delays caused by retransmission, so they build their own mechanisms (for error control, delay adaptation, etc.) over UDP.



HTTP HyperText Transport P.
NNTP Network News Transport P.
FTP File Transfer P.
IRC Internet Relay Chat
SMTP Simple Mail Transfer P.
UDP User Datagram P.
IP Internet P.
ICMP Internet Control Message P.
ARP Address Resolution P.

PPP Point-to-Point Protocol
SDH Synchronous Digital Hierarchy
ATM Asynchronous Transfer Mode
AAL ATM Adaptation Layer
DNS Domain Name Service
SNMP Simple Network Management P.
TCP Transmission Control P.
BGP Border Gateway P.
OSPF Open Shortest Path First

FIGURE 2-5: The traditional protocols in the TCP/IP architecture.

2.3.2 Integrated Services (Intserv)

The purpose of IETF Integrated Services (Intserv or IS) Working Group [intserv] was to specify an enhanced service model to support multiple services in the Internet, and define and

standardize certain interfaces and requirements that are necessary for implementing the service model. The Working Group focused on defining a minimal set of requirements to extend the service model, leaving enhancements to individual protocols to other related Working Groups. An overview can be found in [rfc1633], and more aspects in other RFCs available in [intserv]. The CSZ multiservice architecture [clar92a] is an example of a network architecture based on Intserv.

The view was to provide a new service model for the Internet, evolving from the traditional one, that is, the one that includes the best-effort service and is based on the existing internet layer protocol IP. Four services were defined:

- Guaranteed Service, a guaranteed delay service in deterministic terms.
- The Controlled-Load service, a guaranteed delay service in qualitative terms (low jitter and low loss).
- The Link Sharing service, a hierarchical-sharing service.
- The best-effort service.

Routers must be able to reserve resources (link's capacity, buffer space, etc.) for a given flow. In Intserv routers identify flows and maintain per-flow state, which represents a fundamental change to the traditional Internet model, where per-flow states were only present in the end-systems. Intserv is a core-stateful network (see Subsection 2.2.4) and therefore it has low scalability.

The finest granularity of the model is an (end-to-end) flow from an application that requires certain QoS (aggregation of flows can also be considered). In this case the state is configured dynamically during the call setup. This explicit mechanism requires an "instantaneous" AC and a per-flow signaling protocol (a reservation setup protocol). In other cases, manual configurations or the Simple Network Management Protocol (SNMP) can be used. Traffic conditioning mechanisms are used to ensure that the flow doesn't violate the agreed traffic profile, and administrative controls are used to decide whether the user has administrative permission to request the service. Moreover, the extension is designed from the beginning to consider multicasting.

A reference implementation framework for the Intserv architecture is proposed in [rfc1633]. There are four components, the classifier, the packet scheduler, the AC (all three providing traffic control), and the reservation setup protocol (Figure 2-6):

- The (multifield or MF) classifier finds the flow (or an aggregate) that each arriving packet belongs to (flows are defined in a list), so that the scheduler treats all flows' packets in the same way.
- The packet scheduler manages the forwarding of different packets using a set of queues. The queue discipline can differentiate between flows or aggregates. Traffic conditioning is done at the edge of the network and is considered one of the functions of the scheduler.
- The AC component implements the decision algorithm that a router or host uses to determine whether a new flow (or an aggregate) can be granted the requested QoS without impacting earlier guarantees. AC is hop-by-hop, i.e., it is invoked at each node to make a local acceptance or rejection decision at the time the host requests the service along a path. Not only a yes/no decision can be made, since another possibility is to inform the requesting application about a lower QoS that can be provided.
- The reservation setup protocol is a signaling protocol for creating and maintaining a QoS state for a flow (or an aggregate) in routers and hosts along a path.

The Resource ReSerVation Protocol (RSVP) was the reservation setup protocol chosen for Intserv. It was specified by the IETF Resource ReSerVation Protocol (RSVP) Working Group [rsvp] mainly in [rfc2205], and other RFC's available in [rsvp]. RSVP carry service requests

(with the flow identifier, QoS and traffic parameters) and the corresponding replies from the AC, from source host to router, router to router, and router to destination host (or hosts in the case of multicasting). An important feature of RSVP is that it is receiver-oriented, i.e., the receiver of the flow initiates and maintains the resource reservation used for that flow. Moreover, RSVP reservations are maintained by "soft-states", i.e., each reservation state has an associated timer, and when the timer expires the reservation is removed. If the receiver wants to maintain the reservation state, it must periodically refresh it by sending reservation messages. The receiver can also change the reservation by adjusting these reservation messages.

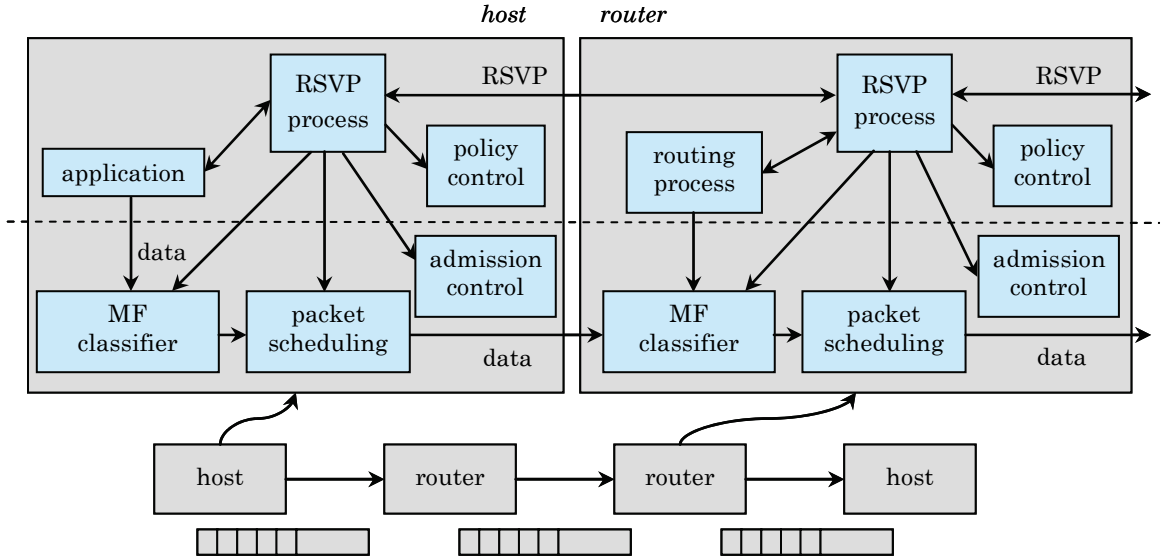


FIGURE 2-6: Functional block diagram of the Integrated Services architecture.

2.3.3 Differentiated Services (Diffserv)

We have just seen that the Intserv architecture proposes a new service model that is able to provide reservation of resources to individual flows. However, some difficulties appear in its deployment due to its core-stateful architecture. Using the per-flow state at each node, hop-by-hop AC, per-flow signaling and even per-flow queuing results in low scalability, which can be a problem in backbone networks that handle a high number of flows. For these reasons, other ways were explored. The IETF Differentiated Services (Diffserv or DS) Working Group [diffserv] guided another architecture to support QoS in the Internet.

Some of the main goals of Diffserv are to achieve high scalability and flexibility. Diffserv scales well to large networks since, in contrast to the per-flow orientation of Intserv, it uses a core-stateless architecture (see Subsection 2.2.4) using packet classes. Packets are classified into a small number of packet classes that receive a particular treatment in queues. Simple functionality is placed in the core network routers, while the more complex operations are implemented at the edge of the network. Moreover, flexibility is needed because new services may appear and old services may become obsolete. Therefore, Diffserv does not define services as does Intserv, but it provides the functional elements of the network with which the services can be built.

Diffserv is more oriented to static configuration of the state, instead of a dynamic one like Intserv. Packets are sent to the network without saying anything about the service and the

network identifies the flow (or usually an aggregation) and provides the agreed service. All the features related to the service delivery are previously defined in SLAs between the customer and the service provider. An administrative management is required to allocate resources for each SLA. As mentioned previously, it is usually a long-term operation and could be done by manual configuration or by using a signaling protocol, but this question has not been addressed by the Diffserv Working Group. The Working Group did not work on new signaling mechanisms, mechanisms for identifying flows, or on defining SLA's. The following are some of the requirements of Diffserv identified by the Working Group [rfc2475]:

– "it should accommodate a wide variety of services and provisioning policies; it should work without the need for API changes or host software modifications; it should not depend on hop-by-hop application signaling; it should avoid per-flow reservation or per-customer state within the core network nodes; it should utilize only aggregated classification states within the core".

The Diffserv architecture is mainly defined in [rfc2474] and [rfc2475], and its origins come from [rfc2638] and [clar98a]. The architecture consists in two sets of functional elements (Figure 2-7):

– At the edge of the network, "sophisticated" packet classification and traffic conditioning is carried out. Packets are assigned to packet classes.
– In the core, a simple classification is needed. Queue disciplines based on the packet classes are applied.

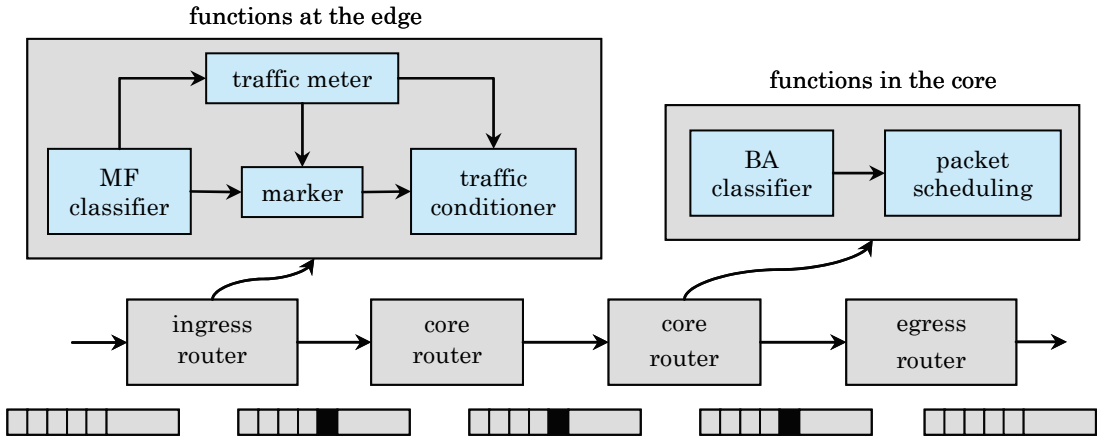


FIGURE 2-7: Functional block diagram of the Differentiated Services architecture.

At the incoming edge of the network (that is, at either a DS-capable host or at the first DS-capable router), the arriving packets are marked through a "sophisticated" (multifield o MF) packet classification (according to some predefined rules in the SLA). The marking consists in setting several bits of the so-called "Differentiated Services field" (DS field) of the packet header (e.g., the Type-Of-Service octet in IPv4) to some values. The mark of the packet identifies the so-called Per-Hop-Behavior (PHB), that is, the forwarding treatment relative to the QoS that the packet will receive from the router. A collection of packets with the same value of the DS field crossing a link in a particular direction (that is, with the same forwarding treatment) is called a Behavior Aggregate (BA) or a DS BA (or simply a "packet class").

The DS field is defined to substitute the earlier definitions of the IPv4 Type-Of-Service field and the IPv6 Traffic Class field [rfc2474]. The DS field occupies bits 0 to 5 (i.e., 6 bits) of those fields and contains the mark or DS Code Point (DSCP). The DSCP codifies the PHB applied to the packet.

After classification and marking, traffic conditioning is applied at the edge of the network. A metering function is performed to measure the temporal properties of the classified flow (or usually an aggregation), and through comparison with the agreed traffic profile, out-profile and in-profile packets are determined. Then:

- Out-profile packets can be shaped, that is, delayed in a queue until they are in-profile, and then forwarded; or they can be policed, that is, discarded; or they can be re-marked, that is, marked with a new "inferior" DSCP and then forwarded; or they can be forwarded unchanged while triggering an accounting procedure.
- In-profile packets are forwarded unchanged, but in some situations they can be re-marked (e.g., if they are entering a Diffserv domain that uses different mapping between PHB and DSCP).

The meter device takes information from a classifier device and acts appropriately upon marker, shaper and dropper devices. Marking of in and out-profile packets and the decision of shaping, dropping, re-marking or immediately forwarding are not specified in the architecture. Diffserv provides a framework and architectural components that are expected to be flexible enough to accommodate a constantly evolving set of services to end users.

The main network core function is to apply the queue discipline in order to provide the agreed PHB to the packet class (or BA). Routers perform simple classification through the DSCP subfield, which is then mapped to a PHB. Remember that the PHB applied is based only on the mark, so that packets with different sources and/or destinations that share some parts of their respective path and that are equally marked, are aggregated and treated in the same way by a router (they belong to a packet class or BA). Therefore, the aggregation means that there is no per-flow state, which is an important point in order to meet the high scalability requirement.

PHB is defined in [rfc2475] as "the externally observable forwarding behavior applied at a DS-compliant node to a DS Behavior Aggregate", as mentioned before, the forwarding treatment relative to the QoS. Packets in the queues can be treated in a few ways, such as "send first" or "drop last". A PHB does not define any implementation mechanisms, buffers or link's capacity allocation policies, but it defines the behavior or per-hop service a packet class (or BA) receives. The extension of this concept, from a single link to edge-to-edge behavior, is the goal of the Per-Domain Behavior (PDB), defined in [rfc3086] as "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".

The Expedited Forwarding (EF) PHB [rfc3246] and the Assured Forwarding (AF) PHB group [rfc2597] were two PHBs that were standardized:

- The EF PHB provides low delay, low jitter and low loss to an EF aggregate, by ensuring it is served at a certain configured rate. Intuitively, the EF behavior is simple. Delay and jitter are minimized when queuing delays are minimized. Therefore, EF packets should encounter short or empty queues. Furthermore, if queues remain short relative to the available buffer space, packet loss is also kept to a minimum. Short or empty queues are achieved if the service rate of EF packets exceeds their arrival rate, independently of other non-EF traffic. EF PHB can be implemented by a single FIFO queue with priority over non-EF traffic, so that the service rate is the link's capacity. Another option is a single FIFO queue with a high weight in a Weighted Fair Queuing scheduler, so that the service rate is the corresponding link's share.
- The AF PHB group provides different levels of forwarding assurances (no loss). There are no delay or jitter requirements. The AF PHB group is composed of several AF packet classes, which in turn are composed of several packet (sub)classes, each with different levels of drop

preference. Four AF packet classes with three levels of drop preference within each AF packet class were defined. Each AF packet class is independent from the others, and is allocated a certain amount of resources (buffer space and link capacity) that conforms the minimum service rate of the AF packet class. The level of forwarding assurance a packet receives depends on the traffic load and the service rate of the AF packet class, and on the drop preference of the packet within the AF packet class. Packets of the same flow that belong to the same AF packet class must not be reordered regardless of their drop preference. An AF implementation must detect and respond to long-term congestion within each AF packet class by dropping packets while handling short-term congestion by queuing packets. This requires Active Queue Management mechanisms that monitor instantaneous congestion and compute a smoothed congestion level (an average queue length) that is used to determine when packets are discarded. An example of this is the RED algorithm [floy93a] and its extension RIO [clar98a] or similar algorithms [feng99b].

Diffserv does not define services, as does Intserv, but it provides the functional elements of the network with which the services can be built, such as the PHBs. Some services have been proposed using its functional blocks. The network service provided to a flow (or an aggregation) is the result of applying traffic conditioning at the edge router and PHBs over the packet classes in the core. EF PHB is intended to build delay services in statistical terms. An example is the Premium Service, which provides low maximum delay, low jitter and low loss [rfc2638, nich98a, rfc2598]. The AF PHB group is intended to build throughput services. An example is Assured Service, a relative throughput service that is able to provide different throughputs to different users during congestion [clar98a]. The recent [rfc4594] provides guidelines for using the Diffserv functional blocks to build different network services.

The Diffserv framework does not include as a requirement any mechanism for managing the domain's resources. However, it is supposed that resource allocation is performed in some way by an entity called Bandwidth Broker (BB), which can have either a distributed or centralized implementation. It would also be responsible for the AC (if there is one), edge node configuration, authentication and authorization, domain policies, and for the interactions with other neighboring domains through the corresponding BB. Neighboring domains would be other providers (Diffserv-based or not) and would have a bilateral agreement based on aggregation of flows.

2.3.4 MultiProtocol Label Switching (MPLS)

The IETF MultiProtocol Label Switching (MPLS) Working Group [mpls] is responsible for standardizing a network architecture based on using logical paths. The traditional network layer in the Internet, which uses the datagram mechanism and simple routing (e.g., based on the shortest path in terms of hops), is not flexible enough to manage the network efficiently. Using logical paths together with better routing techniques allows MPLS networks to be managed in a more efficient way. MPLS is not a multiservice network architecture in the same sense as Intserv or Diffserv, rather it is a base technology that gives support to mechanisms for optimizing network resource utilization and performance, for controlling the QoS and for increasing network reliability [rfc2702]. Nowadays MPLS is widely deployed in networks worldwide.

The core of the MPLS architecture [rfc3031] is the use of logical paths (LPs), a pre-established path through a set of routers that an aggregation of flows travels through. LPs were also used in ATM networks, and were called Virtual Paths, while here they are called Label Switched Paths (LSPs). Instead of the datagram mechanism in IP, MPLS uses the label-swapping paradigm, a mechanism that is similar to the traditional virtual circuit, in order to build LPs. Routers belonging to an MPLS domain are called Label Switching Routers (LSRs). An

aggregation of flows goes from an ingress LSR to an egress LSR through an LSP. Each LSR maintains a table containing forwarding information relative to its (previously established) LSPs. At the network ingress, a label is added to the packet, and then LSRs use this label to read the table and forward the packet to the next LSR in the LSP. The label has a local significance and is changed by each LSR of the LSP. The mechanism works in the following way (Figure 2-8):

– The ingress LSR classifies an incoming packet into a specific Forwarding Equivalent Class (FEC). A FEC is a group of IP packets with certain common properties (protocol, size, origin, destination, etc.), which are forwarded in the same manner over the same network path. A label (according to the FEC) is then added to the packet, and the labeled packet is forwarded to the next LSR in the LSP.
– Each LSR examines the label and uses it as the index for looking up the forwarding table. The incoming label is replaced by the outgoing label (the label-swapping paradigm) and the packet is forwarded to the next LSR in the LSP.
– Before a packet leaves the MPLS domain, the egress LSR removes its MPLS header.

The step before using the LSP (corresponding to a particular FEC) is to establish it. This requires selecting a path using routing procedures, and distributing labels to fill the LSR forwarding tables using a signaling protocol (e.g., RSVP-TE [rfc3209]).

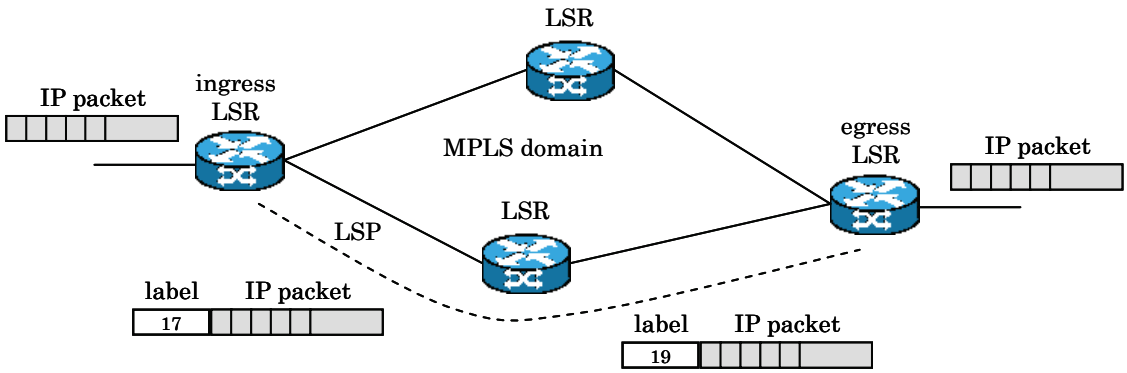


FIGURE 2-8: The basics of the MPLS architecture.

Using LPs together with better routing techniques allows MPLS networks to be managed in a more efficient way than traditional Internet [vila04a]. Optimizing network resource utilization and performance, controlling the QoS and increasing network reliability are some of the management possibilities. The set of LPs between ingress and egress nodes over the physical network results in a logical network through which flow aggregations will travel. A set of resources can be reserved for each LP (e.g., link's capacity), although they can also be established without any resource reservation. The main characteristic of this logical network is its flexibility, since it is possible to change the topology, the resources assigned to each LP, etc., to adapt it to traffic changes and be optimized. Constraint-based routing techniques can be used to setup LPs in longer but lightly loaded paths or based on other criteria, in order to improve resource utilization or to meet policy constraints. Alternative paths between ingress and egress nodes can now be used simultaneously by establishing several LSPs. The management functions can be specialized by creating different LPs for different network services or different users. Using backup LPs facilitates fault restoration in order to increase network reliability. Routing a flow is simplified since it is only done at the network ingress by deciding which LP it is assigned to. Using LPs instead of the traditional datagram mechanism allows pinning a flow to a path, that is, all flows' packets follow the same path, although there

are routing changes. This makes the issue of reserving resources for a flow easier, since in most AC schemes, resources are reserved in a particular path for an accepted flow only at the time the flow is accepted (it is not refreshed); therefore, it is usually assumed that all packets of an accepted flow follow the same path where the reservation has been made. Moreover, if LPs are established with resource reservation, the AC decision is simplified, since it can only be based on local information in the ingress node (no reservation has to be made in transit nodes). The main drawback of using LPs with resource reservation is that it is possible that the LPs passing through the same link share the link's resources inappropriately (some can be congested while others can be underutilized), or that a significant number of LPs divide the link's capacity into small blocks, limiting the possible statistical multiplexing gain.

MPLS is not considered a multiservice network architecture in the same sense as Intserv or Diffserv, since it does not deal with defining network services or mechanisms for building them (scheduling, traffic conditioning or AC). Instead, it is a base technology that can be combined with mechanisms for controlling the QoS, as explained above. Another example is the proposal of using MPLS to support Diffserv [rfc3270].

2.3.5 Intserv, Diffserv and MPLS: a comparison

The IETF proposed two broad architectures to turn the Internet into a multiservice network, firstly Intserv and then Diffserv. The differences between these two architectures are as follows:

- In Intserv, each router, whether at the edge of the network or in the core, has information about all passing flows (e.g., source and destination IP addresses, protocol, source and destination ports). This information can be used through packet classification to treat each flow in a specific way by the queue disciplines, or it can also be used to determine the available resources to perform AC. In Diffserv, per-flow state is only used by edge routers, but not by core routers. Edge routers assign the flows to one of a small set of packet classes, and core routers classify and treat packets based only on the packet class the packet belongs to. The method consists in writing a mark on each packet that identifies the packet class (in the DS field).
- Intserv is able to provide more complex and better services to individual flows; however, using per-flow state at each node and even per-flow queuing results in a low scalability with the number of flows, which can be very large in core routers. Diffserv cannot provide such powerful services but it is highly scalable, so it was considered more appropriate, especially in backbone networks.
- In Intserv, specific end-to-end services are defined, while in Diffserv PHBs are defined (the per-hop service received by a packet class). The end-to-end service in Diffserv is built by applying traffic conditioning to a flow, then marking a packet class, and then applying the corresponding PHBs over the chosen packet class in each router of the path. Each provider creates his own end-to-end services.
- Using AC and per-flow signaling is present from the beginning in Intserv, while in Diffserv it is considered an external element to the architecture. In fact, Diffserv aims at providing services with and with no absolute guarantees, and if AC is used, it must be implemented without per-flow state or per-flow signaling in the core in order to keep the complexity to a minimum.

MPLS is a network architecture based on using LPs, a base technology for giving support to mechanisms for optimizing network resource utilization and performance, and mechanisms for controlling QoS and increasing network reliability. Unlike Intserv and Diffserv, it does not deal with defining network services or with the mechanisms for building them (scheduling, traffic conditioning or AC), but it can give support to these mechanisms. For instance, using

LPs instead of the datagram technique makes resource reservation for a flow easier, since LPs allow pinning the flow to a path. Moreover, if LPs are established with resource reservation, the AC decision is simplified. Another example is the proposal of using MPLS to support Diffserv [rfc3270].

2.4 TCP elastic traffic

In this section we deal with TCP elastic traffic. Firstly, we discuss the QoS requirements of "data" applications, starting from the application QoS and then the network QoS. The users of these applications expect that there is no error in the transfer of documents and also that the response time is the smallest possible below a certain maximum value; therefore, transfers require reliability and the maximum possible throughput above a minimum value. Then we give a general definition of the network service for elastic flows, i.e., a minimum throughput and if possible, an extra throughput, which we call the Minimum Throughput Service (MTS). After that we describe in more detail the two important functions of TCP: reliability through packet retransmission and resource sharing through rate-adaptive algorithms. Finally, we describe the characteristics of TCP elastic traffic at different levels, as seen as a set of sessions, documents, packets and flows, and we summarize the main results about statistical traffic models at these different levels.

2.4.1 QoS for elastic traffic

In "data" applications, the application's processes transfer discrete (time-independent) messages or "documents" (a web request, a basic web file, an embedded image, an ftp file, an ftp command, a typed character in telnet, an e-mail message, etc.). The QoS at the application layer (Subsection 2.1.2) is described in terms of fidelity to the original documents and in terms of interactivity or response time (see Figure 2-9). Fidelity refers to the errors in the transferred documents, while the definition of the response time varies depending on the application. For example, on the web, the response time may be defined as the waiting time between requesting a page (user "click") and visualizing it, which includes the transfer of several documents (the initial request, the basic web file, the rest of the requests, some embedded images, etc.); in ftp, the response time may be defined as the waiting time between commands and status messages, and especially between a file request command and the end of the file transfer; in telnet, the response time may be the time between when a character is typed at the client and the visualization of the corresponding echo sent by the server. In general, the response time is composed of the transfer times of documents and the processing time by the application's processes (e.g., a web server).

Specifically, users of these applications expect no errors in the transfer of documents, i.e., absolute fidelity. Moreover, the smaller the response time the more satisfied the user, but when the response time is too long, impatient users or high layer protocols may interrupt the transfer [mass99a]. These aborted transfers imply a waste of resources, which can get even worse if the transfer is tried again. This means that there is a maximum response time. Its value depends on users' desires and the specific application. For example (see [nour02a] and references therein), a normal user browsing the typical small web pages, expects a maximum response time of a few seconds (e.g., 5 s); in ftp, where files are typically larger, the maximum response times are also larger, and users would be willing to wait in proportion to the file size; or in telnet, the echo delays should be smaller than 150 ms. Moreover, some demanding users can want better performance than others, e.g., users using the web for business applications (e-commerce, online trading, etc.) may require smaller maximum response times than users browsing the web for a "normal" use. In conclusion, the users of these applications expect that

there is no error in the transfer of documents and also that the response time is the smallest possible below a certain maximum value.

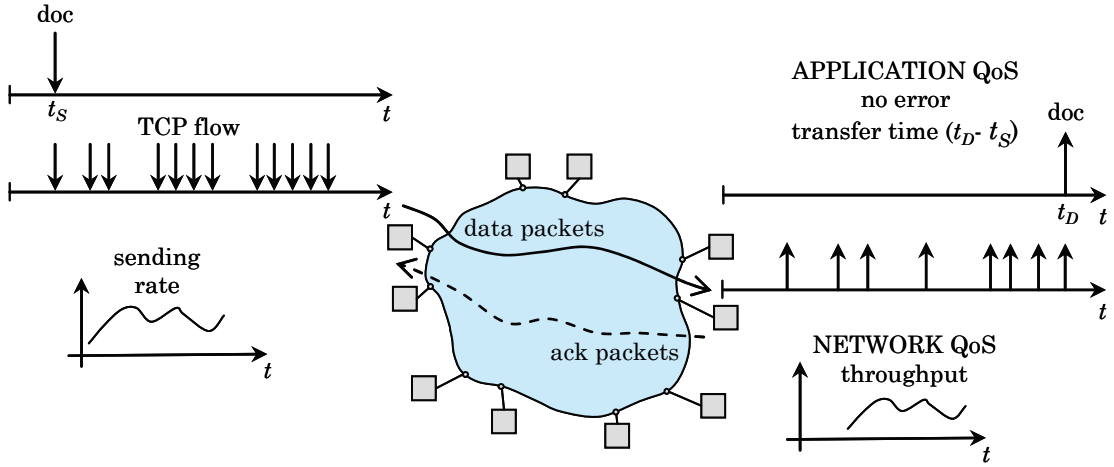


FIGURE 2-9: Application QoS and network QoS in elastic applications.

For transferring an individual document, the above requirements imply that there should be no error and the smallest possible transfer time below a maximum value (see Figure 2-9). Then:

– Absolute fidelity can be achieved through packet retransmission procedures, as in TCP. The TCP source divides the document into blocks (for small documents a single block may be enough) and sends a sequence of packets at a certain sending rate, which the network delivers to the destination occasionally with delays and some losses. From the acknowledgment packets sent back by the destination, the source detects and retransmits lost packets until the whole document is received correctly. Packet retransmission increases the packet delay and consequently the document transfer time, and moreover, it may cause duplicated packets, which are discarded by the destination. From the point of view of the network, the decisive QoS parameter is the average receiving rate (averaged in some time interval) or network throughput, which includes the duplicates. From the point of view of the application, the average receiving rate without including the duplicates is more important, known as goodput. The goodput, if averaged in a period equal to the transfer time, is simply the document size divided by the transfer time.
– The requirement about the document transfer time turns into a requirement about the throughput, i.e., the document transfer should achieve the maximum possible throughput above a minimum value. Note that the traditional view is different since the minimum throughput requirement is not considered. Traditionally, the utility curve of these applications, which relates user's satisfaction to throughput, is considered to be strictly positive and concave [shen95a]. This means that users always benefit by any increase in throughput (i.e., any reduction in the document transfer time) but also that users tolerate throughputs tending to zero (i.e., unlimited document transfer times). However, because users expect a maximum response time, a maximum document transfer time is required, and therefore, a minimum throughput is required. In conclusion, the requirement of the smallest possible document transfer time below a maximum value implies that the network should provide a minimum throughput and if possible, an extra throughput, and also that the source should be able to use it, as in TCP. TCP sources use rate-adaptive algorithms to achieve the maximum possible throughput while sharing network resources fairly between all TCP flows [jaco88a]. Since the maximum possible throughput changes over time, TCP increases and decreases the sending rate in order to match these variations and minimize

packet loss. Due to this ability of adjusting the sending rate to different network conditions, "data" applications and TCP flows are called "elastic".

2.4.2 The Minimum Throughput Service (MTS)

Elastic flows require the maximum possible throughput above a minimum value from the network. Therefore, they are satisfactorily supported by a network service that provides a minimum throughput to the flow and if possible, an extra throughput, which we call the Minimum Throughput Service (MTS).

The input traffic profile of the service is defined by an average sending rate equal to the desired minimum throughput. Flows' packets can be considered to be in-profile or out-profile by comparing the actual average sending rate of the flow and this input traffic profile (see Figure 2-10). Then:

- In-profile packets are delivered, i.e., they have no loss (there are no requirements on packet delay). This results in the minimum throughput.
- Out-profile packets can be delivered, i.e., they can have some loss. This depends on the remaining network resources, that is, the ones that are not used by the in-profile traffic of flows. These remaining network resources are shared between competing flows according to a defined sharing policy, e.g., using best-effort sharing, equal or weighted sharing, giving priority to short flows over long flows, or other. This results in the extra throughput.

Finally, the delivery of the service from the provider to the user (and end-user or a neighboring domain) should be defined in a SLA (see Subsection 2.1.5).

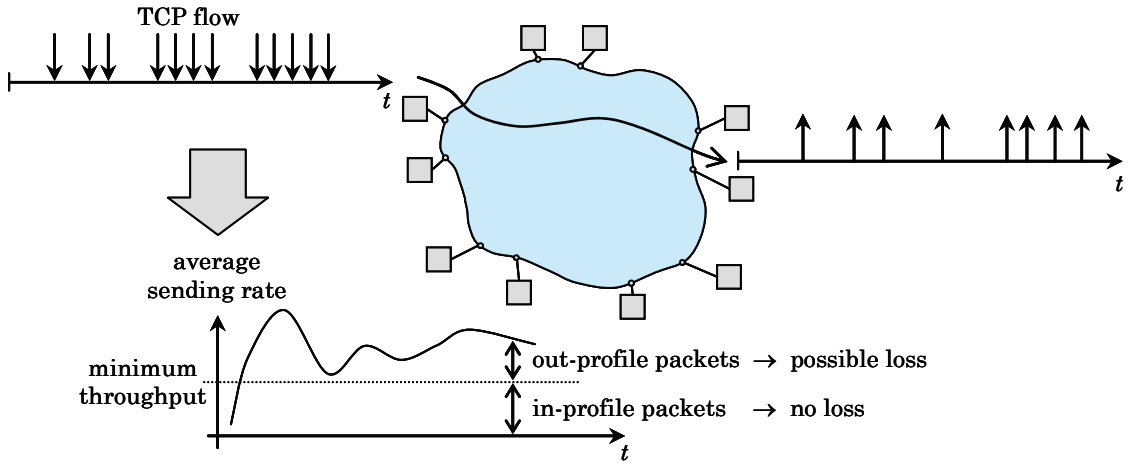


FIGURE 2-10: The definition of the Minimum Throughput Service.

2.4.3 Reliability and resource sharing in TCP

TCP is a transport protocol that provides a connection-oriented, reliable and ordered service to the application layer, besides performing multiplexing of traffic from different application's processes through the ports. The protocol is standardized in [rfc793] but a large number of other RFCs deal with different aspects of TCP. In this subsection we review the two important functions of TCP, reliability and resource sharing.

The following is a summary of how TCP provides a reliable delivery [nour02b, come95a]:

- Application data is partitioned by the source in blocks of at most MSS (Maximum Segment Size) bytes and sent into TCP packets, which carry the (per-byte) sequence number of the first byte of the block.
- The destination only notifies correctly received packets, by sending back Acknowledgment (ACK) packets. ACKs are "cumulative", i.e., they indicate the correct reception of all bytes before the carried (per-byte) sequence number (which is the sequence number of the next expected byte). If an out-of-order packet arrives, a duplicated ACK is sent, and if an in-order packet arrives, the ACK sending may be delayed (but one ACK should be sent for at least every second packet arrival and no later than 500 ms after the first arrival [rfc1122]). Selective ACKs (SACK), which indicate non-contiguous blocks of consecutive bytes correctly received, have also been defined [rfc2018, rfc2883].
- TCP uses pipelining together with the sliding window mechanism for flow control. This combination allows the source to have multiple sent but yet-to-be-acknowledged packets, with a limit equal to the size of the window (new packets are allowed to be sent when ACKs corresponding to previously sent packets are received). The size of the window is the number of bytes the destination can accommodate in its receive buffer (it is called the receiver's advertised window), and it is notified at the source through ACK packets. Typically the resulting traffic is very bursty since the source sends a number of packets continuously (a burst) according to the window and then stops and waits for the ACKs before going on.
- There is a timeout-based mechanism at the source to detect the situations in which no ACK is received for a particular packet. When a packet is sent, a timer is initialized to a time called retransmit timeout (RTO).
- A packet is considered to be lost when the corresponding ACK is not received within the RTO (RTO expiration), or when three duplicated ACKs of a previous ACK are received (this second procedure is called "Fast Retransmit" [rfc2581]). A third possible loss indication comes from the SACK information, if it is used.
- When the loss of a packet is detected, a retransmission procedure is triggered. Depending on the actual state, only the lost packet is retransmitted ("selective repeat" style), or the lost packet and the next packets in the actual window ("go-back-N" style). Packet retransmissions can cause duplicated packets that are discarded by the destination.

It is worth commenting the following about the RTO expiration and the Fast Retransmit procedure, the two traditional indications of packet loss:

- RTO should be longer than the round-trip time (RTT) of packets to allow for ACK arrivals, but not too much so as not to delay retransmissions. RTT can be measured but, since RTT changes in time, it is difficult to estimate its actual "right" value (and consequently the "right" RTO). Since it is desired that the timer expires early only on rare occasions, RTO is obtained through a conservative calculation based on the average and deviation values of measured RTTs [rfc2988] (moreover, before the first RTT measurement has been made, RTO is set to a value of 3 s). On the other hand, the timer is initialized with the actual RTO when a packet is sent or retransmitted. Usually there is a single timer related to the oldest unacknowledged packet, and when this packet is acknowledged, the timer is reinitialized (with the actual RTO) for the next unacknowledged packet. If the timer expires, the actual RTO is doubled ("exponential back off") and the timer is reinitialized. Moreover, TCP implementations use coarse grain clocks to measure the RTT and trigger the RTO. This limits the precision of all these procedures and imposes a large minimum value on RTO. [rfc2988] states, again in a conservative approach to avoid early retransmissions, that whenever RTO is computed, if it is less than 1 second then the RTO should be rounded up to 1 second. The conclusion is that RTO expiration may take a relatively long time.
- The Fast Retransmit procedure [rfc2581] is expected to detect a packet loss before RTO expiration, ideally after a time of about one RTT (sometimes a few RTT), so retransmissions

are faster. However, since three duplicated ACKs are required, at least three later packets have to be sent and correctly received at the destination, and therefore, when the window is small, a late RTO expiration is more likely to occur.

Besides reliability, another important function of TCP is to achieve the maximum possible throughput while sharing network resources between TCP flows. This is the reason why TCP sources use rate-adaptive algorithms.

Resource sharing between TCP flows has the general goal of using the resources fully while maintaining a certain "fairness" in the allocations to flows. Fairness can be defined in different ways, such as max-min fairness, proportional fairness and other (see [mass02a] for a discussion), leading to different allocations. For example, according to the classical fairness notion, the so-called max-min fairness, in a simple scenario of N flows sharing a single link of capacity C, the fair rate for each flow is equal to C/N. In a general network, this does not simply mean allocating the same share to each flow in a link-by-link basis, since this may not lead to full utilization. Then [bert87a]:

- Max-min fairness is achieved when the rates allocated to flows are made as equal and large as possible, or more formally, when an increase in any allocated rate is at the cost of a decrease in some already smaller rate.
- Or alternatively, when each flow has a "bottleneck" link, i.e., a link 1) that is fully utilized, and 2) where the flow's allocated rate is equal to or larger than the rates allocated to the rest of the flows using this link.

Another notion of fairness consists in minimizing the number of actual flows by giving priority to short flows over long flows, which has been shown to reduce the transfer time of short documents without hurting the performance for long flows, when considering heavy tailed document size distributions [mass00a, bans01a].

Another point apart from the fairness type is that the fair rate of a flow changes during its lifetime. This is because the number of flows in the network changes in time, due to new arrivals and departures of finished transfers. Therefore, the average allocated rate of a flow (and the corresponding document transfer time) depends on two issues, the type of fairness and the variations in the number of flows [mass00a].

The fair rate is not explicitly indicated to TCP sources. Instead sources use a probing method that reacts according to binary indications from the network, i.e., whether the sending rate is below the fair rate ("no congestion") or the opposite ("congestion"). The classical congestion indication is packet loss. TCP sources use rate-adaptive algorithms (called congestion control algorithms) that increase the sending rate while there is no congestion, and decrease the sending rate when congestion occurs, oscillating around the fair rate (and adapting to changes in its value). The amplitude of the oscillations (which should be limited to avoid inefficiencies in link utilization) as well as the rate of convergence and adaptation to changes (packet loss should be minimized to reduce retransmissions) depend on the specific rate-adaptive algorithms. Moreover, the network does not enforce the fair rate on the TCP flow, and therefore the fairness in resource sharing is achieved relying on all TCP sources implementing the same algorithms. As a consequence, the type of fairness also depends on the specific algorithms used by all sources [mass02a].

As mentioned above, the algorithms react according to congestion indications from the network, usually packet loss. The following is a more complete summary of possible congestion indications:

- Packet loss detected from Fast Retransmit. This is considered a fast detection method. It does not work well when the window size is small.

– Packet loss detected from RTO expiration. It may take a relatively long time in comparison to Fast Retransmit. This is considered to be an indication of severe congestion, because Fast Retransmit has not detected the packet loss before.
– Packet loss detected from SACK information.
– An increase in RTT. Before queues overflow (and packets are dropped), the RTT of packets increases, and this can be used by TCP sources to react in advance and reduce losses, with the consequent improvement in performance.
– Explicit Congestion Notification (ECN). With ECN [rfc3168], routers can provide an explicit binary indication of congestion to end-nodes before packet loss occurs. Two bits in the IP header are used: one for indicating the congestion and another for indicating the ECN capability. By using Active Queue Management mechanisms such as RED [floy93a], routers set the congestion indication bit in packets when the queue occupancy is high enough but before the queue overflows (and a packet has to be dropped). TCP uses ECN in the following way: when the destination TCP receives a packet with the congestion indication bit set, it echoes back this bit (through one dedicated flag of the TCP header) in its next ACK to the TCP source, which then reacts to congestion as if a single packet loss had occurred. With ECN, TCP performance improves because losses are reduced.

TCP sources vary the sending rate by controlling the window size, because the average sending rate (in RTT) is roughly equal to the window size divided by RTT (this comes from considering that TCP sends a burst of packets limited by the window size and then waits for ACKs before going on, which arrive after one RTT). In this way, the window, now renamed as the congestion window (*cwnd*), can vary from 1 (MSS) to the actual receiver's advertised window.

TCP congestion control algorithms have evolved over time, resulting in the so-called "TCP versions" (see [nour02b] for a general view). The first one, TCP Tahoe [jaco88a], defined the following mechanisms for increasing the window (later standardized in [rfc2581]):

– Slow Start. The window is set to a small value (less than or equal to 2 (MSS), typically 1), and then it is increased by 1 (MSS) after each new (i.e., non-duplicated) ACK is received (*cwnd* = *cwnd*+1). If the receiver acknowledges every packet, *cwnd* is doubled each RTT (a multiplicative increase by 2). When the window reaches a value called "Slow Start threshold" (*ssthresh*), it continues increasing according to Congestion Avoidance.
– Congestion Avoidance. The window is increased as *cwnd* = *cwnd*+(MSS·MSS)/*cwnd*, after each new ACK is received. If the receiver acknowledges every packet, *cwnd* is increased by approximately 1 (MSS) every time a full window is acknowledged, i.e., it is increased by 1 (MSS) each RTT (an additive increase by 1).

When a packet loss is detected, through Fast Retransmit or RTO expiration, *cwnd* is set to 1 (MSS), entering Slow Start, and *ssthresh* is set to *FlightSize*/2 (but no less than 2 MSS), where *FlightSize* is the amount of data that has been sent but not yet acknowledged. A "go-back-N" retransmission procedure is used. Therefore, TCP Tahoe starts from "one" and performs fast probing through Slow Start and slow probing through Congestion Avoidance. When a packet loss is detected, it starts again from "one", and *ssthresh* (which initially can be arbitrarily large, e.g., the receiver window) is adjusted dynamically so that the next slow probing is performed as the window is near the value at which a loss previously occurred. Note also that if delayed ACKs are used, the window is increased at a lower rate since less ACKs are sent.

The second version, TCP Reno [rfc2581], differs from the first one only in terms of its behavior after a Fast Retransmit, which is considered an indication of moderate congestion. The Fast Recovery algorithm was introduced:

– When a packet loss is detected through Fast Retransmit, *ssthresh* is set to *FlightSize*/2 (but no less than 2 MSS), and *cwnd* is set to *ssthresh*+3. The lost packet is retransmitted, and if

allowed by the window, new packets are sent (i.e., "selective repeat" style). For each additional duplicated ACK, *cwnd* is increased by 1 (MSS). When a new ACK is received, *cwnd* is set to the actual *ssthresh* (i.e., the previous *FlightSize*/2, a multiplicative decrease by 2), and it enters Congestion Avoidance.

However, it was shown that this procedure, by requiring every packet loss to be retransmitted strictly based on Fast Retransmit, may fail to recover from multiple losses in a single flight of packets, which leads to RTO expiration for the other lost packets. An improvement of Fast Recovery was introduced in a new version, TCP NewReno [rfc3782], which was extensively used. Basically, during Fast Recovery, "partial" ACKs (new ACKs not covering the highest sequence number sent) and "full" ACKs are distinguished: if a partial ACK is received, the next corresponding packet is considered to be lost and retransmitted, and if a "full" ACK is received, then Fast Recovery ends. Another way of dealing with the problem of multiple losses in a single flight of packets is using the SACK option, since its information can be used to selectively retransmit the lost packets. A modified TCP Reno with SACK was shown to outperform TCP NewReno in [fall96a], especially when the number of losses is large. The SACK option is widely deployed but not widely used, although straightforward implementations have been proposed [rfc3517].

We have just seen above the classical increases and decreases of the TCP sending rate to achieve the fair rate (see Figure 2-11): when losses do not occur and when in Congestion Avoidance, the window is additively increased by one (*cwnd*+1) each RTT, and when losses occur and are detected through Fast Retransmit and recovered through Fast Recovery, the window is multiplicatively decreased by two (*cwnd*/2). This is known as a particular case of the more general "Additive Increase and Multiplicative Decrease (AIMD)" control behavior.

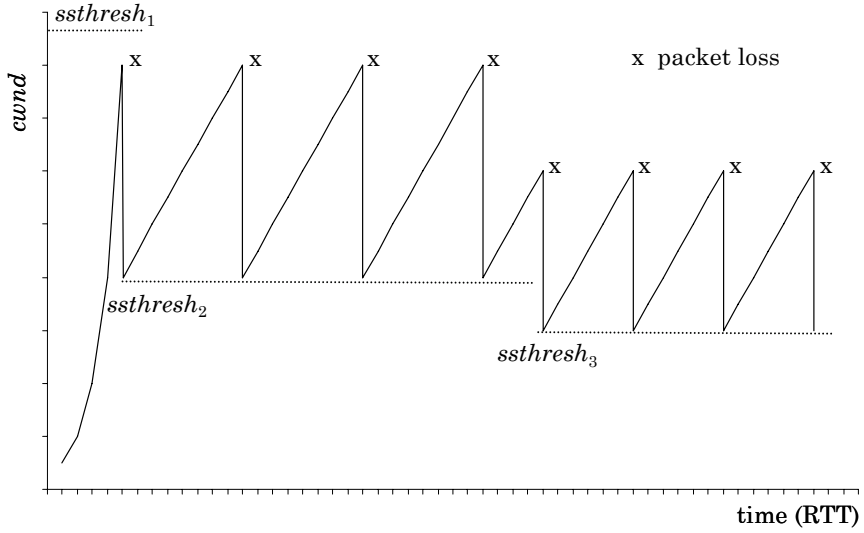


FIGURE 2-11: The ideal AIMD (Additive Increase Multiplicative Decrease) behavior of the TCP congestion window, after an initial Slow Start phase.

AIMD was studied in a single link in [chiu89a], which showed that it converges to fair resource sharing. Although it is often stated that AIMD rate variations provide max-min fairness in a general network, some authors (e.g., [mass02a]) have shown that they tend to provide rather another type of fairness called "proportional fairness" (which produces smaller allocations for flows passing through more hops to the advantage of greater overall throughput). Moreover, fairness in resource sharing between TCP flows depends strongly on the RTT and time duration of flows:

– Flows with large RTT achieve smaller throughput than flows with small RTT. This is because the value of the additive increase of the sending rate (the congestion window) is constant and independent of RTT, and it does not occur at fixed time intervals but in time periods of RTT due to the necessary feedback delay [floy92a]. Therefore, the sending rate increases more quickly for flows with a smaller RTT, and achieves higher throughput. This happens with long flows, which are stable in Congestion Avoidance under AIMD control.
– Short flows tend to achieve smaller throughput than long flows. This is because short flows spend most of their lifetime in Slow Start, while long flows spend most of their lifetime in Congestion Avoidance, and flows in the Slow Start phase achieve smaller throughput than flows in the Congestion Avoidance phase. Firstly, flows during Slow Start double the sending rate each RTT (until a loss is detected), but meanwhile they achieve a lower throughput since it is necessary to start conservatively from a small value. Secondly, during Slow Start the window may be small, and if losses occur it is probable that they will be detected through RTO expiration and not through Fast Retransmit (as it has been observed in measurements, e.g., in [bala98a]). The expiration may take a long time (RTO is usually large, since there are just a few samples of RTT at the beginning, a conservatively value is used), and moreover, the window is severely decreased due to starting again from a small value in Slow Start. Thirdly, flows in Congestion Avoidance have larger windows and are less sensitive to losses (usually detected through Fast Retransmit), and are stable under AIMD control around the fair rate.

Several analytical models have been developed to predict TCP performance. The steady state throughput of a long flow (a "bulk transfer") is modeled in [padh98a], considering the flow's RTT, Fast Retransmit and RTO expiration as loss indications, the limitation of the congestion window size to the receiver's advertised window, and that packet losses (with probability p) within a single flight of packets are correlated (but independent of other flights). Although the formula for the long-term throughput is more complex, it can be approximated as $K/(\mathrm{RTT}\sqrt{p})$ packets/s, as long as the packet loss rate is not too high and the receiver window is not limiting (K is a constant equal to $\sqrt{3/4}$ or $\sqrt{3/2}$, depending on whether delayed ACKs are used or not respectively). This formula confirms the RTT unfairness between long flows we described above. This model is extended in [card00a] by adding the connection establishment and the Slow Start phase, in order to predict the performance of short flows as well.

Another important aspect in TCP resource sharing, the effect of the number of flows changing in time, is studied in [ben01b] (and extended in [bona03b]). Its analytical model statistically characterizes the throughput of a random number of flows that share resources fairly. The model considers a single link of capacity C, where flows arrive according to a Poisson process (with λ arrivals/s), carrying a document with a size according to any statistical distribution (with mean σ [bit]). Moreover, it is considered a rate limit for the flows $r < C$ (to model the limit due to the receiver window size or other bottleneck links), and perfect and instant adjustments of the sending rate in response to changes in the number of flows (an ideal fair resource sharing) are assumed. The performance is shown to depend only on the average traffic load $\lambda\sigma$ and C, being stable if $\lambda\sigma < C$, and insensitive to the specific size distribution. The statistical distribution of the number of flows in progress is derived, as well as the mean throughput (approximately the minimum between C–$\lambda\sigma$ and r). These basic results were obtained considering Poisson flow arrivals, and were shown to be valid for the more realistic traffic assumption of Poisson session arrivals in [bona01d].

Finally it is worth commenting that the evolution of the TCP congestion control algorithms does not end with the classical algorithms we have seen above. There have been many other modifications and proposals for "new" TCPs in order to improve its performance, some of which we briefly describe here. As we have already stated above, TCP can benefit from adding ECN to IP [rfc3168], as well as using Active Queue Management such as RED [floy93a] instead of the classical Tail Drop. There are also some proposed TCP versions that use the variations in

RTT as a congestion indication (e.g., TCP Vegas [brak95a], Fast TCP [jin05a]). TCP Westwood [case02a] estimates the available throughput by monitoring the rate of ACK packets, and uses it to better adjust the congestion window. A very different proposal is XCP [kata02a], which is based on explicit congestion information from routers carried in packets that indicate how much the congestion window values have to be varied. In TCP pacing schemes (e.g., [agga00a]), in order to reduce TCP traffic burstiness, the source evenly spreads the sending of a window of packets across an entire RTT (at an average rate of *cwnd*/RTT), instead of sending a burst. TCP ACK pacing schemes (e.g. [awey02a]) regulate how quickly the congestion window increases by controlling the rate of arrival of ACK to the TCP source. There are some proposals that aim to improve the performance in high-speed and long distance networks, in which classical TCP underutilizes the available throughput (HighSpeed TCP [rfc3649], scalable TCP [kell03a]), and also to solve the RTT unfairness problem (BIC-TCP [xu04a], Fast TCP). Other proposals aim to improve the performance in wireless networks, in which packet losses may not be caused by buffer overflow, but rather by a relatively high bit error rate, and therefore, TCP should not reduce the sending rate (problems and solutions are surveyed in [tian05a]). Some other proposals are classified and compared in [lai01b], and numerous references can be found in Sally Floyd's homepage [floyd].

2.4.4 Characteristics of TCP elastic traffic

Traditional "data" applications in the Internet generate the majority of Internet traffic. This is confirmed by traffic measurements (see for example, [thom97a, brow02a, fral03a]), where TCP is by far the dominant protocol and other protocols, such as UDP, contribute to a much lesser degree to the total traffic. Within TCP traffic, the dominant application is the web, although lately the traffic generated by peer-to-peer file sharing applications has also become an important part [fral03a]. The rest of TCP traffic mainly comes from the classical ftp and e-mail applications.

Traffic can be described at different levels by considering different entities as a set of sessions, documents, packets and flows. The notion of session generally refers to a time period of "continuous" and "related" user activity, so that user sessions can be considered statistically independent. A session has a starting time and duration, and is composed of a succession of documents. Documents are generated within a session, and are characterized by its sending time and its size. Each document results in a sequence of packets, each packet is characterized by its sending time and length. A flow is a sequence of "related" packets that are "close" in time, which can correspond to the transfer of a single document, several documents or an entire session. It is characterized by its starting time, duration, traffic parameters such as the average rate, peak rate, etc., document size, and others. The sequence of packets corresponding to a document transfer (which includes the retransmitted packets) is typically very bursty (the TCP source sends a number of packets continuously – a burst – according to the actual window and then stops and waits for the ACKs before going on) and has a variable average rate (due to the TCP rate-adaptive algorithms). Moreover, besides data packets, control packets for connection management and error correction are also sent.

The structure of sessions, in terms of documents, their interarrival times and sizes, and the TCP connections used, depends on the application. The following is a qualitative description of some applications:

– During a web session in a server, the user downloads a set of web pages, each composed of several parts called "objects", usually a "basic" file and several embedded images (referenced in the basic file). A user "click" results in the basic file being requested, and once it is received, the client requests the rest of the objects of the web page. A TCP connection can be of two types, a "non-persistent" connection, when it is closed by the server after finishing the transfer of an object, or a "persistent" connection, when it remains opened and is closed by

the client or the server usually when there is inactivity during a given timeout interval. The set of documents may be transferred [rfc1945, rfc2616] within several non-persistent TCP connections (one connection per document, opened sequentially or in parallel), or within a single or several persistent TCP connections (each one with sequential pairs of request-reply, or with "pipelined" requests, that is, several requests one after the other without waiting for each reply). The size of web requests usually fits in a single TCP packet, while the size of replies is extremely variable, since they can range from small basic files to very large files. The web also creates document interarrival times that are very variable. For example, very short interarrival times occur when clients open parallel TCP connections to transfer several embedded images of a web page; short interarrival times come from users browsing and reading different web pages; users taking a long break results in long interarrival times.

– During an ftp session, the commands sent by the client and the corresponding status messages from the server are transferred within a single TCP connection (for "control"). A separate TCP connection (for "data") is established each time the user wants to transfer some data, for example, listing a directory or getting a file (two operations that usually occur close in time). Control commands are small, while the size of the files is extremely variable.
– During a telnet session, there is a single TCP connection, in which each character being typed by the user at the client is sent to the server, which echoes them back, as well as sending the responses to the commands. The size of the typed characters is obviously very small and their interarrival times are limited by the typing speed of the user.

Measuring traffic at the session or flow level may be difficult because, as we have just seen, the relation between these traffic entities and TCP connections is not obvious. In some cases (e.g., in ftp or telnet), a session can be simply equated to an entire single TCP connection, initiated by the connection request packets and ended by the corresponding release packets. On other occasions (e.g., in web), a session may include several and related TCP connections (persistent or non-persistent) and considered to be finished when there is no user activity during a given timeout period. Similarly, a flow can correspond to a sequence of packets within a single TCP connection, within several connections or an entire TCP connection. Usually a flow is identified by a 5-tuple in IPv4 (protocol, source and destination IP addresses and ports), initiated when the first packet arrives and finished when there are no more packets during a given timeout period [thom97a, brow02a, fral03a]. Another option is to equate a flow to an entire TCP connection.

By analyzing measurements in the Internet, statistical models for describing the traffic at different levels have been proposed. The following is a summary of the main results:

– Session arrivals can be considered to follow a Poisson process. This has been observed in Internet traffic measurements for some applications, such as ftp or telnet [paxs95a]. This observation is not surprising and can be generalized to any application, since a Poisson process is known to result from the superposition of independent actions undertaken by a large population of users, each of low relative intensity [bona01d, feld00a]. As described above, the statistical properties within a session are clearly complex and depend on the application, with a high variability in the number of documents, interarrival times and sizes, and presumably presenting some correlation.
– Files sizes in the web have shown to exhibit a distribution with a heavy tail [crov97a]. This means that there is a high variability in sizes, and that most web files are small but a few of them are very large (and consequently, the same is valid for the lifetime of flows, when each one corresponds to a single file: most of the flows are short and a few of them are very long). A reasonable fit to the form of the heavy tail is provided by the Pareto distribution.

- Packet arrivals are well modeled using self-similar processes, as it has been observed in Internet traffic measurements [paxs95a, crov97a]. This means that time series of traffic presents high variability or bursts (extended periods above the mean) that remain at a wide range of time scales (bursts are not smoothed when the timescale varies). Traffic also presents long-range dependence, i.e., it shows significant correlations across large time scales. Self-similarity seems to be caused by the heavy-tailed distributions of file sizes [crov97a, park96a].
- The TCP connection arrival process has shown self-similar behavior, and its interarrival times can be modeled by distributions with heavy tails, especially the Weibull distribution [feld00a]. This model can also be considered valid for flow arrivals, although flows and TCP connections are not exactly the same. The model means that the arrival process is bursty, and the variability of interarrival times is high. However, as stated in [bona03b], it may be appropriate on certain occasions to suppose flows arrive according to a Poisson process, as for example, when flows correspond to a large number of sessions and the spacing of flows within a session is large compared to the average inter-flow interval. Moreover, the basic results obtained under certain conditions (perfect resource sharing by TCP flows) and considering Poisson flow arrivals, have been shown to be valid for the more realistic traffic assumption of Poisson session arrivals [bona01d].

Chapter 3:

Admission Control (AC) schemes in the Internet

In this chapter we study the main AC schemes that have been proposed for the Internet. Firstly we deal with general issues about the AC mechanism and present a classification of AC schemes. Then we describe specific AC schemes in the Internet following this classification. In this description we focus on studying the simplicity of the schemes in terms of how many nodes participate in the AC, the required state, the use of signaling, etc.

3.1 The AC mechanism

In this section we firstly describe the role of AC in the network, how it works and its general goals. After that we present a classification of AC schemes based mainly on the nodes that participate in the AC decisions, i.e., centralized schemes, hop-by-hop schemes, one-hop schemes on LPs with reservation and edge-to-edge schemes. Finally we deal with the concepts of the AC algorithm, reservations based on state or on occupancy and explicit and implicit AC and its relation to signaling.

3.1.1 Introduction to AC

The AC mechanism is an efficient way of dealing with congestion situations in a network. Using AC and a "normal" provisioning of network resources, when congestion occurs, some of the flows receive the requested service with absolute guarantees (they are "accepted") and the rest do not (they are "rejected" or "blocked"). Congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques, but if congestion occurs, AC achieves efficient use of network resources by maximizing the number of satisfied flows. However, using AC complicates the network scheme, and therefore a major concern is making the AC as simple as possible.

The AC procedure is the following. A new flow arriving to the network edge wishes to receive a given network service with absolute guarantees, expressed in deterministic, statistical or qualitative terms (Figure 3-1). Here the term "network" can refer to either an entire network, where source and destination nodes would be hosts, or a single domain within a network,

where source and destination nodes would represent edge nodes of neighboring domains. The requested service comprises a description of the QoS and a description of the input traffic profile that receives this QoS. Network routing finds a path through the network towards the flow's destination. AC evaluates if it is possible in the chosen path to provide the service requested by the new flow while maintaining the service already promised to the actual flows in the path (note that these actual flows may travel, in general, through different network paths, which at some point, share one or more links with the path that has been chosen for the new flow). The AC decision must take into account the QoS requirements and traffic characteristics of the new flow and the actual flows in the chosen path, an information that is distributed in the network. If the flow is accepted, it will receive the desired service during its lifetime, since the necessary resources of each node of the path are reserved for the flow. Moreover, in the case of acceptance and in order to isolate (or protect) the rest of accepted flows, the network should check the flow's input traffic and enforce the agreed traffic profile (e.g., discarding non-compliant packets, degrading them to a lower QoS, etc.), so that if the flow exceeds the traffic profile, it does not damage well-behaved flows. This operation can be performed through traffic conditioning mechanisms and/or some specific queue disciplines (e.g., WFQ), at the ingress node or at each node. It also requires maintaining a list of accepted flows.

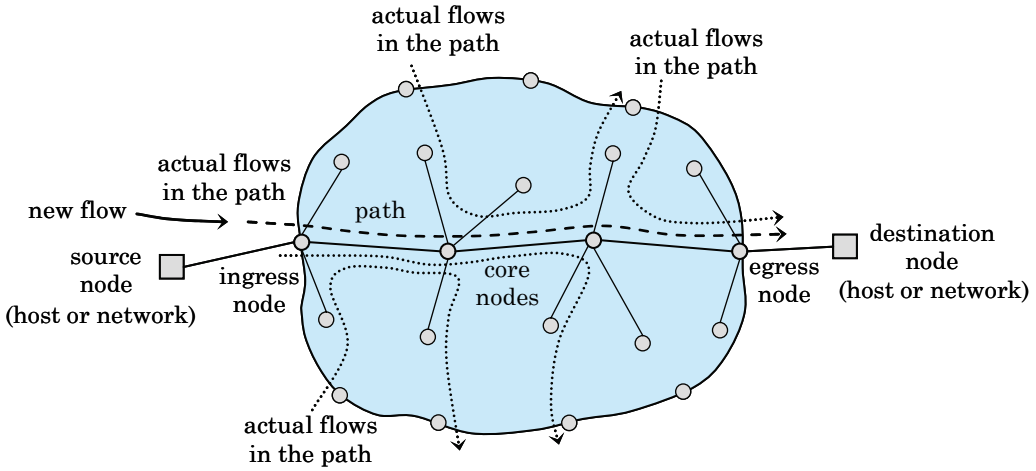


FIGURE 3-1: The AC evaluates if the service requested by a new flow can be provided while maintaining the service promised to the actual flows in the path.

Although the main goal of AC is to maximize the number of satisfied flows, there are other important aspects to take into account when evaluating it, such as the simplicity of the scheme, the duration of the AC phase and others. An AC scheme can be evaluated through the following aspects:

- The ability to provide the desired QoS guarantees (e.g., delay, jitter, loss, throughput), by comparing the target value of the QoS parameter with the average or percentile values of the QoS parameter achieved.
- The efficiency in the use of resources, by comparing the satisfied traffic load with the maximum possible traffic load (e.g., the capacity of links, a desired percentage, etc.).
- The simplicity of the scheme regarding how many nodes participate in the AC, the required state, the use of signaling, the intrusive traffic, computational efforts, etc.
- The duration of the AC phase, i.e., the time that is required for making the AC decision. This should be short since usually it is the time an accepted flow has to wait before starting to transmit and also in order to improve resource utilization.

– The fairness in the AC decisions, i.e., whether different kinds of flows (e.g., flows requesting different traffic rates, traveling through different paths, etc.) have the same likelihood of being accepted.

3.1.2 A classification of AC schemes

We classify the AC schemes according to which nodes are aware of AC, i.e., which nodes participate in the AC decision (Figure 3-2). A basic classification is the differentiation between centralized and distributed approaches:

– In centralized AC schemes there is a single AC entity in the network that makes the AC decisions and exchanges signaling packets with the ingress nodes when new flows arrive.
– In distributed AC schemes, AC decisions are made in different nodes. Distributed schemes can be further classified into several types. In hop-by-hop AC schemes the AC decision is split into partial decisions in each node of the path and each one performs a local AC decision. In one-hop AC schemes for LPs with reservation, a LP from ingress to egress nodes is previously established with a resource reservation, and then only the ingress node is involved in the AC decisions. In edge-to-edge AC schemes, the ingress and egress nodes participate in the AC, since the AC decision is taken from measurements (of different kinds) performed at the egress and sent back to the ingress through special packets.

In centralized AC schemes the AC requests are processed serially, while in distributed AC schemes, they may be processed simultaneously in different nodes. This concurrency may cause false acceptance and rejection decisions. For example, in hop-by-hop AC schemes, concurrency does not lead to false acceptances since reservation requests are processed serially in each node; however, partial reservations that further along the path are rejected, may lead to false rejections of other flows. Other distributed AC schemes also suffer from false AC decisions for similar reasons. False rejections due to many flows arriving simultaneously and causing low utilization are also known as the "thrashing" problem [mitz96a, bres00b].

Our complete classification of AC schemes is shown in Figure 3-2. In the next sections of this chapter we follow this classification to describe specific AC schemes that have been proposed for the Internet.

3.1.3 The AC algorithm, the reservations and the signaling

The central issue in an AC scheme is how to decide to accept or reject a flow and making the corresponding reservation. A false acceptance will cause the violation of the QoS guarantee, and a false rejection a lower utilization. It is a difficult task to understand the interaction between the new flow and the actual flows in the set of queues and links of the path in order to predict if the QoS that all flows will receive will meet the requested QoS. Moreover, the information about the QoS requirements and traffic profiles of the actual flows in all links of the path is distributed in the network and changes in time due to the arrival and departure of flows. This interaction can be modeled using complex theoretical principles or simpler ones, but it results in the so-called AC algorithm, which is a test for making the AC decision.

The AC algorithm may be implemented in one or more nodes depending on the type of AC scheme. It is implemented in all nodes in hop-by-hop AC schemes; only at the edge nodes in one-hop AC schemes on LPs with reservation and also in edge-to-edge AC schemes; and in a single entity in the network in centralized AC schemes.

The scope of the AC algorithm is the portion of the path where the resulting AC decision is valid. It can be a single link or the whole path:

– An AC algorithm with a link scope means that the resulting AC decision is valid in a link. The AC algorithm is used consecutively in each link of the path in order to obtain the total AC decision. For example, in hop-by-hop AC schemes, the AC algorithm is implemented in all nodes, each node maintains the information about the available resources in its links, and usually signaling between nodes propagates the partial decisions in order to obtain the total AC decision. A link-scope AC algorithm may also be used in a centralized AC scheme, where a single entity maintains the information about the available resources in each of the links of the path.
– An AC algorithm with an edge-to-edge (or end-to-end) scope means that the resulting AC decision is valid in an edge-to-edge (or end-to-end) path. All edge-to-edge AC schemes use these AC algorithms. For example, in active measurement-based edge-to-edge AC schemes with per-flow probing, a probing flow with similar characteristics to the new flow is generated and sent through the path, its packet loss is measured at the egress of the path and a special packet carries the measurement to the ingress: the AC algorithm implemented at the ingress compares this measurement with the desired packet loss target in order to decide the admission. Another example is the AC algorithm implemented at the ingress in one-hop AC schemes on LPs with reservation: it takes an AC decision for the whole path as if there were a single link (one hop) from ingress to egress with the LP's reserved resources.

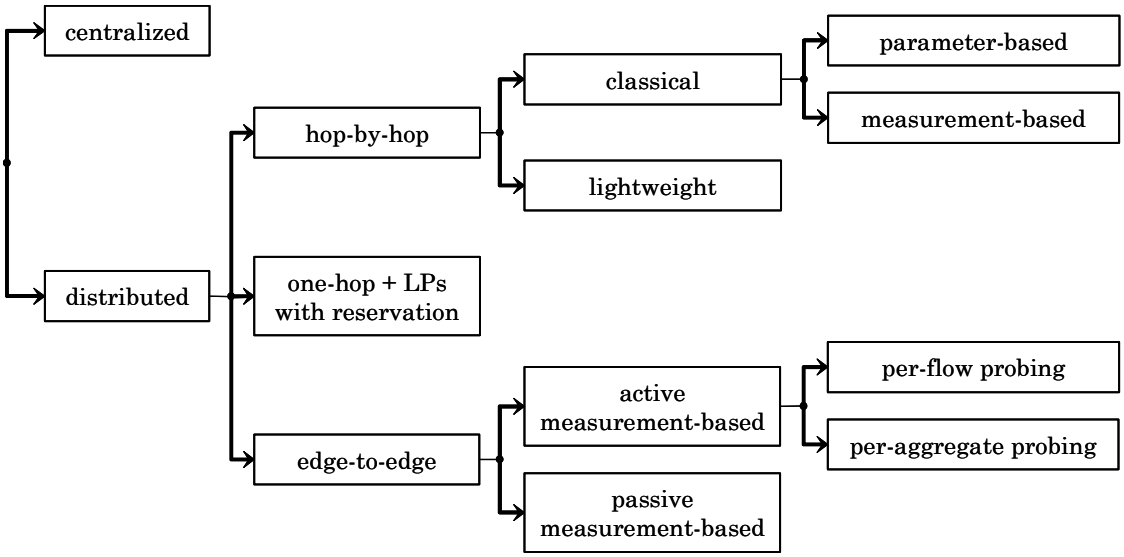


FIGURE 3-2: Classification of AC schemes.

The AC algorithm may use traffic and QoS parameters declared by flows at the admission time, measured traffic and QoS parameters over individual flows or aggregates, resource parameters (queue length, link's capacity), and others. Usually the new flow is characterized by declared parameters at the admission time, but the already accepted flows (or in other words, the aggregated reservation, or reversely, the available resources) can be characterized by its declared parameters or by measurements over the actual traffic. In this way, AC algorithms can also be classified as being parameter-based or measurement-based:

– Parameter-Based AC (PBAC) algorithms use declared parameters for the new and the accepted flows. For example, a link scope algorithm that guarantees no loss and admits a new flow in a link if the sum of all peak rates of the accepted flows plus the peak rate of the new flow is less than the link's capacity.
– Measurement-Based AC (MBAC) algorithms use declared parameters only for the new flow and characterize the accepted flows through measurements. Measurements can be made

over individual flows or over aggregates. They can be passive (i.e., over data packets) or active (i.e., over a special flow that is generated and sent for measuring purposes). An example is a link scope algorithm that guarantees low jitter and low loss and that admits a new flow in a link if the measured average rate of the accepted flows plus the peak rate of the new flow is below a certain percentage of the link's capacity. Another example is the edge-to-edge scope algorithm of active measurement-based edge-to-edge AC schemes with per-flow probing described above, which admits the new flow if the measured packet loss of the probing flow at the egress of the path is smaller than the desired packet loss target.

Once a flow has been accepted, the necessary resources of each node of the path are reserved for the flow. Flows' reservations in a node can be based on state or occupancy:

- State-based reservations are maintained independently of whether the flow transmits or not and only depend on the state. There is a list of the flows that have been accepted in the node, which contains the flows' identifiers and service parameters (and maybe others). There is a link scope and parameter-based AC algorithm for this node, which uses the information of this list.
- Occupancy-based reservations are maintained as long as the flow transmits during its lifetime. There is no list of the flows that have been accepted in the node, and there is either a link scope AC algorithm for this node or an edge-to-edge scope AC algorithm for the path, which use passive or active measurements over individual flows or aggregates. Usually it is required that the flow always transmits following a certain traffic profile that fills the reservation (but "virtual" measurements – Subsection 3.3.3 – can avoid this requirement).

Note that if flows' reservations in a node are based on state, accepting a flow implies a new entrance in its list of accepted flows. For flows' reservations in a node based on occupancy this operation is obviously not necessary. However, in some other node, e.g., in the ingress node of the path, in a centralized entity, etc., a list of the accepted flows should be maintained in order to be able to differentiate between the packets of new flows (i.e., its start and therefore to initiate the AC phase) and the packets of accepted flows, and also in order to isolate (or protect) the accepted flows by enforcing the agreed traffic profile on their input traffic. A flow may be identified by the usual 5-tuple in IPv4 (protocol, source and destination IP addresses and ports) or the more flexible 3-tuple in IPv6 (flow label, source and destination IP addresses).

The distributed nature of any AC scheme implies using some type of communication between the different nodes involved, i.e., the ones where the AC algorithm is implemented, or all the nodes of the path or other, which depends on the type of AC scheme:

- In centralized AC schemes, the centralized entity exchanges signaling packets with the ingress nodes and, if traffic measurements are used or maybe for other reasons, also with all the nodes in the path. The centralized entity maintains the reservations in nodes based on state or occupancy.
- In hop-by-hop AC schemes all the nodes of the path communicate with each other and each node maintains its flows' reservations based on state or occupancy.
- In one-hop AC schemes on LPs with reservation only the ingress node is involved and maintains its flows' reservations based on state or on occupancy (although the previous reservation in the establishment of the LP can involve either all the nodes of the path or a centralized entity and is based on state).
- In edge-to-edge AC schemes only the ingress and egress nodes exchange special packets (e.g., for carrying measurements) and each node of the path maintains flows' reservations based on occupancy.

Therefore, an important issue is how the source, the destination and all the involved network nodes communicate to each other the information that is required for making the AC decision

and (in the case of acceptance) for establishing the reservation, its maintenance and release. Specifically, we refer to communicating information such as the following: the start of a new flow and its requested service parameters (traffic and QoS parameters) in order to begin the AC phase (i.e., the AC request); acceptance or rejection (i.e., the AC response); the end of (an accepted) flow in order to release the reservation (i.e., the AC release request); or traffic measurements, if used. This communication can be made explicitly or implicitly:

- The explicit way uses signaling packets, i.e., extra packets besides data packets. It allows rich and dynamic information to be indicated but it complicates the network and consumes resources that could be used for data traffic.
- The implicit way only uses data packets and signaling packets are not used. It is simpler and extra traffic is avoided, but the range of information is narrower: either the information is the same for all flows (and does not need to be indicated) or it has very few possibilities that can be indicated using an existing field in packets' headers.

Several explicit and implicit procedures can be used:

- The start of a new flow can be indicated by an initial signaling packet, or implicitly, by the first data packet of the flow. This initiates the AC phase. Both packets carry the flow's identifier.
- The QoS and traffic parameters can be indicated in different ways: written in an initial signaling packet; or, for the traffic parameters, indicated by an initial sequence of (signaling or data) packets through the number of packets sent or by its rate; or the first data packet indicates the "type" of service simply through the port numbers or other packet fields marked for that purpose (e.g., a "real-time" or "elastic" mark), and the specific QoS and traffic parameters are not indicated since they are the same for all flows of the same type.
- The acceptance and rejection decision can be indicated by a signaling packet back, or implicitly, by forwarding the data packet in the case of acceptance and discarding it or modifying an agreed mark in the case of rejection. The acceptance of a flow implies updating the list of accepted flows wherever it is placed.
- The end of a flow can be indicated by a signaling packet, or implicitly, when no packet is received within some defined timeout interval. State-based reservations are maintained independently of whether the flow transmits or not. They can be released through a release signaling packet ("hard" state), although a timeout procedure for detecting long periods of inactivity can be used (e.g., for failures); or if a reservation refreshing procedure is used, when the refreshing signaling packet is not received within a defined timeout interval ("soft" state). Occupancy-based reservations are maintained as long as the flow transmits (following a certain traffic profile) during its lifetime. They are released when the flow ends and is inactive, without needing to signal packets (because the related AC algorithm is based on measurements).

3.2 Centralized AC schemes

In centralized schemes there is a single AC entity (where the AC algorithm is implemented) that maintains the information concerning the network topology and the state of resource usage. When a new flow requires admission, a signaling packet is sent from the ingress node to the centralized entity specifying the flow's identifier and the service parameters, then the AC decision is made. For example, if a link scope and parameter-based AC algorithm is used, the entity obtains the available resources in each of the links of the chosen path from its state information database, and using the AC algorithm in each link, makes the AC decision. Then the ingress node is notified through signaling packets (and the AC phase ends). If the flow is accepted, the state information database is updated and nodes in the path are configured as necessary through signaling packets (e.g., for classification, scheduling or traffic conditioning).

If measurements were used in the AC algorithm, signaling packets carry this information from the nodes to the centralized entity.

The early proposals of Bandwidth Brokers (BBs) in Diffserv networks are an example of centralized AC schemes [rfc2638, teit99a]. BBs are responsible for authenticating and authorizing the service request, implementing domain policies about service usage, controlling domain agreements, making the AC decision, and the consequent edge node configuration (for classification and traffic conditioning). In addition, if the flow's destination was outside the domain, BB is responsible for negotiating with the downstream domain's BB (through interdomain signaling) in order to achieve the end-to-end service. A centralized approach was chosen due to its simplicity and because it does not require a state in the core, a major concern in Diffserv.

Centralized schemes allow AC to be performed without complicating the control plane inside the network and eliminating the need for maintaining the distributed state. Moreover, service requests can be processed serially and AC decisions are never made simultaneously. However, the centralized entity has to store the complete network state and process signaling packets in order to update it. This can be difficult to achieve in networks with a large number of flows and highly dynamic behavior (i.e., with many service requests and reservation release events). Moreover, the centralized entity can become a sending and receiving point for a lot of signaling traffic and suffer congestion. Therefore, the centralized scheme is more appropriate in networks in which most flows are long, and service requests and reservation release events are not frequent. Finally, its dependence on a single entity makes the centralized scheme highly vulnerable to failures.

The next sections are devoted to distributed AC schemes, in which AC decisions are made in different nodes and the state information is distributed in the network. Distributed schemes are able to cope with large and highly dynamic networks and are more robust. However, decisions are not made serially as in centralized schemes, and therefore distributed schemes have to face the problems associated with simultaneous AC decisions.

3.3 Distributed hop-by-hop AC schemes

In hop-by hop schemes the total AC decision is split into partial decisions in each node of the path, and each node maintains the actual aggregated reservation and performs a local AC decision (through a link scope AC algorithm). The procedure is the following: if the flow is accepted in a node, a local reservation is made; then this partial acceptance decision is propagated to the next node in the path, and so on; if it reaches the end, the procedure stops and the total AC decision is acceptance; if the flow is rejected in a node, the procedure stops and the total AC decision is rejection. Usually signaling between nodes is used to propagate the partial decisions and to communicate the response: an AC request packet carrying the service parameters is sent from node to node and then an AC response packet carrying the total acceptance or rejection decision is sent back. The duration of the AC phase is about one RTT. Traditional AC schemes are hop-by-hop and use per-flow signaling. The Intserv architecture [rfc1633] adopted the hop-by-hop approach and RSVP [rfc2205] as the signaling protocol (see Subsection 2.3.2).

As a distributed scheme, the hop-by-hop approach can handle several AC decisions simultaneously, which may lead to concurrency problems. Since flows' requests in each node are processed serially, and in the case of acceptance, a local reservation is made, concurrency does not lead to false acceptance. However, "thrashing" may occur [mitz96a]: partial reservations in a hop, made for flows that later are rejected in other hops, may prevent other flows from being accepted in this hop, leading to false rejection.

Another feature of hop-by-hop AC schemes is that they exhibit multihop and multirate unfairness, i.e., they discriminate against flows crossing longer paths (in terms of the number of hops) and flows demanding larger traffic rates [jami97a, bres99a, cost242]. This problem, which is also a problem of other types of AC schemes, happens because admitting a flow is only based on the criterion of whether there are enough available resources. Multihop and multirate unfairness is usually considered to be a question that is orthogonal to the original AC problem, which could be solved through adding other criteria to the AC decision (see, e.g., the "trunk reservation" mechanism in [cost242]).

As we explained in Subsection 3.1.3, the AC algorithm that is implemented at each node can characterize the accepted flows by its declared parameters at the admission time or through measurements. In the next subsections we study in more detail hop-by-hop schemes with PBAC algorithms and with MBAC algorithms.

3.3.1 Classical parameter-based hop-by-hop schemes

Hop-by-hop schemes with PBAC algorithms use declared parameters at the admission time for the new flow and also for the accepted flows (for the aggregated reservation). PBAC algorithms allow building services with deterministic or statistical guarantees:

– One example of a deterministic guarantee is the peak rate allocation algorithm that guarantees zero packet loss if the sum of the peak rates of all accepted flows plus the peak rate of the new flow is less than the link's capacity (note that "real" peak rates vary as flows travel along the path, and therefore it is better to allocate slightly higher peak rates). Another example is the algorithm in [pare93a], which guarantees no loss and a maximum delay D (as the definition of the Guaranteed Service of Intserv – see Subsection 2.3.2 –), by using a WFQ scheduler and characterizing flows with the average rate r and burst size b of a token bucket traffic profile. The flow is accepted if it can be assigned an unreserved portion R of the link's capacity (through the appropriate WFQ's weight), so that $D = b/R$ and $R > r$. Due to the deterministic guarantee, both algorithms are based on considering the worst-case scenario; therefore, if the flows are bursty, the utilization is low.
– Using strong mathematical models, PBAC algorithms can also provide statistical guarantees. By relaxing the deterministic guarantee to a statistical one, these algorithms consider scenarios that are likely to happen in most cases (with certain probability), instead of the worst-case, and thus they achieve better utilization. In [cost242] several AC algorithms (for real-time traffic) are discussed, and classified as either based on rate envelope ("bufferless") multiplexing or on rate sharing ("buffered") multiplexing. In addition, in [knig99a] the performance of several sets of PBAC algorithms is compared: average and peak rate combinatorics, additive effective bandwidths, engineering the "loss curve", maximum variance approaches, and refinements of effective bandwidths using large deviations theory. Similar work is carried out in [perr96a] for ATM networks.

A problem in PBAC algorithms is the need to accurately describe the flow's traffic parameters. It is difficult for a user to tightly characterize its traffic in advance; therefore, the traffic parameters declared at the admission time are inevitably loose upper bounds. This can result in low utilization.

Classical hop-by-hop schemes with PBAC algorithms use reservations based on state (with per-flow state in each node) and per-flow signaling between nodes. The ingress node (or all nodes) isolates the accepted flows by enforcing the agreed traffic profiles to their input traffic. State-based reservations are maintained, no matter whether the flow is active or not, until a release signaling packet is received, or in the case of using a reservation refreshing procedure, until the refreshing signaling packet is not received within some defined timeout interval. The signaling protocol includes an AC request packet that carries the flow's identifier and the

service parameters, an AC response packet that carries the total acceptance or rejection decision in the path, and either an AC release packet or an AC refreshing packet. The procedure is the following:

- Upon receiving the AC request packet, each node performs a partial AC decision.
- If the node accepts the flow, it makes a local reservation, it forwards the AC request packet to the next node, and it waits for the AC response packet that carries the total AC decision; if the destination accepts the flow, an AC response packet of acceptance is sent back to the source through the same path, and partial reservations are confirmed. The duration of the AC phase is about one RTT, after which an accepted flow is allowed to transmit.
- If the flow is rejected in one node, an AC response packet of rejection is sent back to the source through the same path, and partial reservations are released.
- When the flow ends, either the AC release packet is sent through the path or the periodic sending of the AC refreshing packet is stopped, and the reservations in the path are released.

Note that these schemes require the per-flow state to be maintained in each node. In principle, it is possible to envision a simple solution without needing identifying individual flows, where each node maintains only the actual aggregated reservation, which is updated for each flow's acceptance or departure indicated by the signaling packets. However, if signaling packets suffer a loss, these packets are then retransmitted, and this may generate duplicates that nodes have to be able to detect by maintaining the per-flow state (to avoid duplicated reservations or releases). Moreover, per-flow state is also required in order that each node knows the previous node in the path followed by the flow, and signaling packets can be sent back through this path. The need to maintain the per-flow state in each node and to process per-flow signaling packets to update it, results in scalability limitations; therefore, this is one of the main disadvantages of these AC schemes.

3.3.2 Classical measurement-based hop-by-hop schemes

Hop-by-hop schemes with MBAC algorithms use declared parameters at the admission time for the new flow and characterize the accepted flows (the aggregated reservation) through measurements. Measurements are (passively) taken over aggregates during a certain time interval and the estimated parameter is usually the average rate (the traffic load) or others (e.g., the delay, loss, etc.). The estimation obtained from measurements in a given time interval is used to make AC decisions in future time intervals. Usually, when a flow is accepted, the estimation is updated at once (artificially) to take into account the decrease in available resources, because measurements won't reflect this until the next time interval.

In MBAC algorithms, besides the AC algorithm itself, there is a second component, the measurement process, which can be carried out in different ways. As an example, we describe the Measured Sum algorithm together with the Time Window measurement mechanism, for a low jitter and low loss service [jami97b]:

- The Measured Sum algorithm admits a new flow if the sum of the rate requested by the new flow and the estimated traffic load of accepted flows is less than some defined percentage of the link's capacity (the percentage is required to bound jitter and loss).
- In the Time Window mechanism there is a measurement window T divided into several time intervals S. At the beginning of T there is an estimated traffic load. For each interval S, the traffic load is averaged. If a new average is above the estimate, the estimate is immediately increased until this new average. If a flow is admitted, the estimate is also immediately increased artificially with the requested rate of the new flow. At the end of T, the highest load average from the T that has just ended is used as the load estimate for the next T.

MBAC algorithms have several advantages over PBAC algorithms. Measurements can better estimate the aggregate behavior in flow's multiplexing and improve resource utilization (especially when the number of flows is high). It also eliminates the problem of tightly characterizing the traffic parameters of the new flow in advance. A very conservative and simple traffic specification (such as the peak rate) does not imply a waste of resources, because after some time, measurements in the next intervals will reflect the actual load. However, using measurements is not easy. Due to the dynamic behavior of flows, measurements taken in past time intervals cannot be accurate predictors for the future (especially in the presence of long-range dependence), and this can cause eventual QoS degradations and even false acceptances. In order to avoid false acceptances, measurements are used carefully, adopting very conservative methods. Due to eventual QoS degradations, MBAC algorithms cannot provide deterministic or even statistical guarantees, but they can provide qualitative guarantees (e.g., a low jitter and low loss service for real-time traffic, similar to the definition of the Controlled Load Service of Intserv, see Subsection 2.3.2).

Examples of MBAC algorithms can be found in [jami97a, qiu01a, gros03a, jama03a], a comparison of a set of algorithms in [bres00a, jami97b, bres99a], and also in [shio99a] for ATM networks. More specifically, the works in [jami97b, bres00a] describe and compare several algorithms (Measured Sum, Hoeffding Bounds, Tangent at Peak, Aggregate Traffic Envelopes, etc.) and several measurement mechanisms of the traffic load (Time Window, Exponential Averaging, and Point Sample), to provide a low jitter, low loss service. One of the conclusions in [bres00a] is that choosing one or another algorithm is not very important, since all achieve nearly identical performance in terms of the loss-load curve (i.e., the loss rate that occurs at a given utilization). The authors suggest that what is more important in the performance is the way the departure and arrival of flows is treated, and the responsiveness of the load measurement to traffic fluctuations:

– When a new flow is admitted, the actual load measurement does not reflect the presence of the new flow yet. In order to prevent the risk of admitting too many flows before the measurement is updated in the following time intervals, all these algorithms take the worst-case approach of increasing the actual load measurement artificially with the declared flow's traffic parameters (e.g., the peak rate). This can reduce the utilization. When a flow ends, the algorithm does not know how much the departing flow was contributing to the actual measurement. The measurement is not explicitly adjusted, and it will not reflect the new situation until the next intervals. This again may reduce the utilization.
– Although the number of flows does not change, the traffic load fluctuates in the sort-term and the measurements reflect this. A high fluctuation can be confused by having a lot of admitted flows, which leads to too many flows being rejected; a low fluctuation can be confused by having few admitted flows, which leads to too many flows being accepted.

The length of the measurement interval can control the above two factors, but with conflicting goals: a longer measurement interval reduces the effect of the short-term fluctuations but it also slows down the reaction of the algorithm to the departure and arrival of flows. Moreover, another important conclusion in [bres00a] is that none of the studied algorithms is able to reliably achieve a loss rate close to the targeted loss rates.

In hop-by-hop schemes with MBAC algorithms reservations are based on occupancy due to the measurements being used. Classical schemes use per-flow signaling between nodes but per-flow state in the core is not required. Per-flow state is only kept at the ingress in order to detect new flows and to isolate the accepted flows by enforcing the agreed traffic profile on their input traffic. Occupancy-based reservations are maintained as long as the flow transmits (following a certain agreed traffic profile), and they are released when the flow ends and is inactive, without needing an AC release packet. The signaling protocol includes an AC request packet that carries the flow's identifier and the service parameters, and an AC response

packet that carries the total acceptance or rejection decision in the path. The procedure is the following:

- Upon receiving the AC request packet, each node performs a partial AC decision.
- If the node accepts the flow, it makes a local reservation (i.e., the measurement is artificially increased, e.g., with the flow's peak rate), and it forwards the AC request packet to the next node (the node does not need to wait for the AC response packet since in any case the aggregated reservation is updated through measurements); if the destination accepts the flow, an AC response packet of acceptance is sent back to the ingress node (and then to the source) to confirm the flow in the list of accepted flows. The duration of the AC phase is about one RTT, after which an accepted flow is allowed to transmit.
- If the flow is rejected in one node, an AC response packet of rejection is sent back to the ingress node (and then to the source), and the flow is erased from the list of accepted flows. Partial reservations are released in future measurements.
- When the flow ends, the reservations in the path are released in future measurements.

Note that the effect of possible duplicated AC request packets will also be corrected by future measurements.

3.3.3 Lightweight hop-by-hop schemes

These schemes are hop-by-hop since each node maintains the actual aggregated reservation and performs a local AC decision. However, they differ from the classical schemes in the two previous subsections in one or more aspects of the common goal of being simpler. These aspects are the following:

- Some schemes use less per-flow signaling between nodes, or even no signaling.
- The majority of schemes do not require a per-flow state in the core (only at the edge).
- All schemes use (passive) measurements over aggregates. However, some use "virtual" measurements instead of the "real" measurements used by classical MBAC algorithms. "Virtual" measurements can be seen as a way of using PBAC algorithms but without maintaining per-flow state in the node.

By "virtual" measurements we mean that the traffic load is not directly measured from ordinary data packets but from some "special" (data or signaling) packets that reflect the flow's reserved rate (and not the actual one, which can be smaller). In consequence, the aggregated reservation provided through "virtual" measurements aims to be equal to the sum of the reserved rates of the flows. This is in contrast to the one provided through "real" measurements, which aims to be the actual aggregated rate taking into account the multiplexing gain. Reservations maintained through "real" measurements require that the flow always transmits following a certain agreed traffic profile that fills the reservation, while through "virtual" measurements this is not required. "Real" measurements may increase resource utilization but run the risk that traffic fluctuations may cause eventual QoS degradations; therefore, they provide qualitative guarantees. "Virtual" measurements may provide deterministic or statistical guarantees but probably at the cost of reducing resource utilization. Therefore, "virtual" measurements obtain a similar behavior to PBAC algorithms without requiring per-flow state.

The Scalable resource Reservation Protocol (SRP) scheme [alme98a] provides a low jitter and low loss service for real-time traffic (qualitative guarantees). It does not provide isolation to accepted flows and the scheme relies on the cooperation between sources implementing the same algorithms. These are the basic features of the data path:

- There are three types of packets: "requested", "reserved" and "best-effort" (the real-time service coexists with the best-effort service). All of these packets are flow data packets. In

each node, "reserved" and "requested" packets are scheduled with priority over "best-effort" packets.
– When a flow is accepted, the source sends data packets marked as "reserved" at the accepted rate.

The AC scheme of SRP uses "real" measurements, and per-flow state is not required in the network core or at the edge. There is no AC request packet and the only signaling used is an AC response packet from the destination to the source (e.g., using RTCP):

– A source that wishes to make a reservation starts by sending its first data packets marked as "requested". The rate of these packets is the requested rate of the service. The source estimates the requested rate by measuring the sent "requested" packets.
– Each "requested" packet is forwarded unchanged by network nodes if it is accepted, or degraded to "best-effort" if it is rejected.
– The destination estimates the accepted rate by measuring the received "requested" packets, and then it sends this information back to the source through a signaling packet (the final accepted rate is the minimum between the source's measured requested rate and the destination's measured accepted rate).
– The local AC decision tries to keep the load of "reserved" and "requested" packets below a dedicated portion of the link's capacity so that they are not discarded and therefore have a low delay. At the beginning of a certain time period, each node knows the actual reserved load for this period (and therefore the available resources). An arriving "requested" packet is accepted only if enough resources are available, and if so, the amount of available resources is accordingly reduced. During the same time period the node monitors the arriving "reserved" and "requested" (and accepted) packets to determine the new reserved load for the future intervals.

The Load Control scheme [tura01a, marq01a] provides a low jitter and low loss service for real-time traffic (qualitative guarantees). It provides isolation to accepted flows. The basic features of the data path are the following:

– There are four types of packets: "probe", "marked", "refresh" and "ordinary". In each node, "probe", "refresh" and "ordinary" packets together receive a higher priority in queues than "marked" packets.
– When a flow is accepted, the ingress node marks (in-profile) flow's packets as "ordinary". It also refreshes the reservation periodically by marking some of these packets (or generating packets, if necessary) as "refresh". One "refresh" packet in a given refreshment period indicates the reservation of one unit of resources for the time of this period.

The AC scheme of Load Control uses "virtual" measurements, per-flow state is not required in the core and signaling is not reduced:

– A basic requested rate is defined in the domain, which requires one "unit of resources" (e.g., based on averages or effective bandwidths). When a flow desires a given number of unitary resources, the ingress node sends the same number of "probe" (signaling) packets through the path.
– A "probe" packet is forwarded unchanged by network nodes if it is accepted, or changed to a "marked" packet if it is rejected.
– When the "probe" or "marked" packets reach the egress node, they are sent back to the ingress node, which makes the AC decision (these backward packets can also be used to probe the backward path if necessary).
– For the local AC decision, each node knows, at the beginning of a given refreshment period, the amount of reserved resources (and the ones that are available). An arriving "probe" packet is accepted only if one unit of resources is available, and if so, the amount of available resources is reduced by one unit. During this refreshment period the node

estimates the actual load to determine the amount of reserved resources for the next refreshment period. The measurement takes into account the accepted "probe" packets and the "refresh" packets (instead of the "ordinary" packets). Finally, some variations of this scheme were proposed in [marq01a], such as replacing the initial sequence of "probe" (signaling) packets by a single signaling packet carrying the desired resources, using an AC release packet and improving the measurement algorithm.

The scheme proposed in [stoi99a] provides a service with deterministic delay bound and no loss for real-time traffic. It provides isolation to accepted flows. A basic point of the scheme is the queue discipline:

- It uses the Core Jitter Virtual Clock scheduler, which provides the same deterministic delay bound as a set of WFQ schedulers. Per-flow state is not maintained in nodes, it is carried by data packets using a technique called Dynamic Packet State (DPS): the ingress node initializes several state variables encoded in the packet's header; in all nodes the scheduler processes each incoming packet based on this state, and then updates both its internal state and the state in the packet's header before forwarding it to the next node; the egress node extracts the state.
- When a flow is accepted, the ingress node writes each packet's "virtual" length in its header, i.e., the number of bits that the flow was supposed to transmit at its reserved rate r since the previous packet was transmitted. It also inserts in the packet's header the initial values of the state variables required for the scheduling.

The AC scheme of [stoi99a] uses "virtual" measurements, per-flow state is not required in the core and signaling is not reduced:

- When a new flow arrives to the network edge, the ingress node sends a request signaling packet for a reservation of peak rate r through the path. If a node in the path accepts the flow, it forwards the request packet to the next node; otherwise it sends back a reject signaling packet to the ingress.
- For the local AC decision, each node knows the actual aggregated reservation at the beginning of each measurement period, *Rbound*. A new flow is accepted if *Rbound* plus the peak rate r is below the link's capacity, and then *Rbound* is increased in r. During each measurement period, the node also calculates *Rcal*: it measures the actual aggregated rate using the "virtual" lengths instead of the real ones; and moreover, each time a flow is accepted, *Rcal* is also increased in r. At the end of the measurement period, *Rbound* is updated to the minimum value between *Rbound* and *Rcal*.

The scheme proposed in [siva00c, siva99b] provides a minimum throughput service with deterministic guarantees for elastic traffic. It provides isolation to accepted flows. The basic features of the data path are the following:

- When a flow is accepted, the ingress node turns some (in-profile) flow's data packets into "markers". The "marker" carries a number of data packets (or bytes) that it represents. The rate of "markers" indicates the accepted minimum throughput r. In each node, ordinary and "marker" packets are scheduled together using FIFO.

The AC scheme in [siva00c, siva99b] uses "virtual" measurements, per-flow state is not required in the core and signaling is not reduced:

- When a new flow arrives to the network edge, the ingress node sends a request signaling packet for a reservation of rate r through the path. If a node in the path accepts the flow, it forwards the request packet to the next node; otherwise, it sends back a reject signaling packet to the ingress.
- For the local AC decision, each node knows, at the beginning of a given time period, the available bandwidth B_{av} for this period. A request is accepted if r is available ($r < B_{av}$), and then B_{av} is reduced by r. During this time period, the node estimates the value of B_{av} to be

used for the next time period from the number of "markers" received over that time (taking into account the number of packets that the "markers" represent).

The scheme proposed in [kuma00a, mort00a] provides a minimum throughput service with qualitative guarantees for elastic traffic. The minimum throughput's value is the same for all flows and a flow is a TCP connection. It does not provide isolation to accepted flows and the scheme relies on traditional cooperation between TCP sources implementing the same algorithms. The data path is simply based on FIFO queues. The AC scheme uses "real" measurements, per-flow state is not required in the core or at the edge and there is no signaling:

– The start of the flow is indicated to the node through the TCP connection establishment packets (SYN or SYN/ACK). In the case of acceptance, the connection establishment is allowed to proceed by forwarding these packets; otherwise, the connection establishment is aborted.
– For the local AC decision, the node measures a particular parameter (the occupancy of the link in [kuma00a] and the incoming traffic to the link's queue in [mort00a]), and when it exceeds a given threshold, new connections are rejected. The relation between the threshold and the minimum throughput comes from analytical models of TCP connections sharing a single link.

The scheme proposed in [robe04b, kort04a] provides two services with qualitative guarantees, a low jitter and low loss service for real-time flows and a minimum throughput service for elastic flows. The minimum throughput's value is the same for all elastic flows while the peak rate of real-time flows should be smaller than a given value. It provides isolation to accepted flows. The basic features of the data path are the following:

– The queue discipline uses the Priority Fair Queuing (PFQ) algorithm, which requires per-flow state in each node. It shares the link's capacity fairly between all flows and also gives scheduling priority to flows whose peak rate is less than the current link's fair rate. In this way, the requested flow's QoS (real-time or elastic) can be implicitly indicated: a flow whose peak rate is smaller than the fair rate is considered a real-time flow; otherwise it is considered an elastic flow.

The AC scheme uses "real" measurements, per-flow state is required in each node and there is no signaling:

– Per-flow state in each node is required to detect new flows and is maintained using an implicit method. A new flow is indicated by the arrival of its first packet. The node indicates a local acceptance decision of the flow by forwarding this packet or a local rejection decision by discarding it. The end of the flow is detected when no packet is received within a defined timeout interval.
– The AC algorithm does not distinguish between elastic and real-time flows. Moreover, the traffic parameter of the new arriving flow is supposed to be a given value (the maximum of the minimum throughput of elastic flows and the possible peak rates of real-time flows). The AC algorithm ensures that the current priority traffic load is smaller than a given percentage of the link's capacity, and that the fair rate is higher than a given threshold, which is chosen higher than the peak rate of the envisaged real-time flows (so that they receive scheduling priority in PFQ queues).

3.4 Distributed one-hop AC schemes on logical paths (LPs) with reservation

Consider a network using LPs from ingress to egress points, established with a resource reservation (e.g., a given capacity). The view is as if there were a single link from ingress to egress with these assigned resources, where an aggregation of flows travels through. In this

way, AC only has to be done locally, only at the ingress, using any ("link scope") AC algorithm in only one hop and only taking into account the LP's reserved resources. Examples of networks supporting LPs with reservation are ATM and MPLS (see Subsection 2.3.4).

Obviously, previously establishing the LP's reservation requires performing AC in the chosen path as if it were a "flow", with its traffic and QoS parameters. Note that unlike the per-flow AC we are considering in this chapter, this AC is usually applied at a higher timescale than the flow's timescale, and with higher traffic requests. A centralized scheme or a hop-by-hop scheme can be used, with a link scope and parameter-based AC algorithm and state-based reservations (that are maintained although there is no traffic). The requested traffic and QoS parameters correspond to the aggregated traffic of the future accepted flows and their QoS requirements. For example, if a network provides a low jitter, low loss service for real-time traffic, implemented using priority queuing plus a limitation of the real-time traffic's load to a percentage of the links' capacity, it must be assured that the sum of the LP's capacity carrying this traffic in any link is below this limitation.

The main advantage of these schemes is their simplicity. It is a local decision, and it is not required to know about flows in other paths. Moreover, per-flow signaling is not necessary between nodes, since no per-flow reservation has to be made in transit nodes. The AC decision is fast since it is made as soon as the flow arrives. Simultaneous AC decisions in different ingress nodes cannot cause any false acceptance since resources in links have been reserved in advance. Thrashing cannot occur since in each LP, decisions are made serially. However, the main disadvantage of these schemes is that in some situations false rejections may occur: it is possible that the LPs passing through the same link share resources in a non-appropriate way, i.e., some might be congested and rejecting flows while others might be underutilized. In this situation the underutilized LP should decrease its capacity and the congested LP should increase it. The modification of the LPs is carried out by other network management functions [vila04a] (which also consider other goals such as the optimizing resource usage, fairness, etc.). Another disadvantage is that a significant number of LPs can divide the link's capacity into small blocks, therefore limiting the possible statistical multiplexing gain.

3.5 Distributed edge-to-edge AC schemes

In edge-to-edge AC schemes only the ingress and egress nodes participate in the AC decision. There is an edge-to-edge scope AC algorithm implemented at the ingress or egress node, which is based on measurements performed at the egress and sent back to the ingress through special packets (usually signaling packets). The nodes of the path do not maintain either per-flow or aggregate reservation state (they are not aware of AC), do not exchange signaling packets and maintain flow's reservations based on occupancy. Per-flow state is only kept at the ingress in order to detect new flows and to isolate the accepted flows by enforcing the agreed traffic profile to their input traffic. These schemes can be classified using active or passive measurements:

- In active measurement-based edge-to-edge AC schemes, a special probing flow is generated and sent through the path, and its QoS is measured at the egress to be used in the AC decision. Moreover, probing can be per-flow, when there is a probing flow for each new flow, or be per-aggregate, when there is a single and continuous probing flow in relation to the aggregation of accepted flows.
- In passive measurement-based edge-to-edge AC schemes, the QoS of the aggregation of accepted flows is continuously measured at the egress to be used in the AC decision.

3.5.1 Active measurement-based edge-to-edge schemes with per-flow probing

These schemes, also known as endpoint AC or end-to-end measurement-based AC (EMBAC), are the majority of active measurement-based edge-to-edge AC schemes. In some of them, the source and destination hosts, instead of the edge nodes, participate in the AC. They have in common that the AC phase consists in generating and sending a probing flow through the path, and then measuring its QoS at the egress to be used in the AC decision. The general procedure is the following:

- For each new flow, a special probing flow travels from the sender to the receiver through the network path. The probing flow is a sequence of extra (i.e., signaling) packets with similar characteristics to the data packet flow.
- The receiver measures the QoS experienced by the probing flow during a defined time interval. The measured statistics can be simply the average rate received during the time interval, or counting the number of received packets with congestion marks (using Explicit Congestion Notification, ECN [rfc3168]), or include more complex jitter statistics.
- The receiver reports the measured QoS through a signaling packet to the sender, which performs the AC decision, or alternatively, the receiver decides and then informs the sender. There is a timeout mechanism in the sender associated with the start of the AC phase to deal with the loss of feedback signaling packets.
- According to the AC decision, the source sends packets or not. The duration of the AC phase is about one RTT plus the measuring time, after which an accepted flow is allowed to transmit.

In some schemes the packets from probing flows are treated inside the network in the same way as packets from accepted flows (in-band probing). In others they are treated differently (out-of-band probing). In out-of-band probing, two packet classes are used: one for the packets of accepted flows and another for the packets from probing flows; in core nodes, accepted packets receive priority when using resources in order to be protected from the effect of probing packets.

Flow's reservations in nodes are based on occupancy. The probing flow establishes the reservation in the node if resources are available just by transmitting; the reservation is maintained as long as the flow transmits (following a certain traffic profile) during its lifetime; it is released when the flow ends and is inactive, without needing to signal packets. As they are based on traffic measurements, these schemes provide qualitative guarantees (e.g., a low jitter, low loss service for real-time traffic) because the dynamic behavior of traffic may cause eventual QoS degradations.

Examples of these schemes can be found in [borg99b, borg99c, bian00a, bres00b], where they are used to build a low jitter, low loss service for real-time traffic. The Phantom Circuit Protocol [borg99b, borg99c] uses a Constant Bit Rate (CBR) probing flow with a rate at 20% higher than the peak-rate of the new flow, and its jitter is measured and compared to a threshold to make the AC decision. It is an out-of-band probing scheme, with two packet classes, the 1st one (low priority) for the probing packets, and the 2nd one (high priority) for packets of accepted flows, and queues in nodes use two-priority scheduling. In [bian00a] an analytical model is developed to evaluate the performance of a similar scheme using a throughput measurement (counting the number of received bytes within the measurement interval) instead of measuring the jitter. In [bres00b] the performance of several schemes based on throughput measurements is discussed in detail through simulation. The schemes differ in using out-of-band or in-band probing, and a dropping or a marking (through ECN) mechanism. Again two packet classes with different priorities are used, and nodes use priority queuing with a rate limited to a percentage of the link's capacity. Using marking and out-of-band probing has been shown to achieve better performance than the other options.

The main advantage of these schemes compared to hop-by-hop AC schemes is the edge-to-edge architecture. They do not require changes in the network core, where reservation state is not maintained (neither per-flow nor per-aggregate), and only edge nodes are AC aware. However, the duration of the AC phase (and also the time an accepted flow waits for transmitting) may be quite long. [borg99c] recommends including 50 to 100 samples in the measurement duration, so that the duration is determined by the slowest flow but it is in the order of seconds. Even so, using a single, reduced measurement interval for an individual (probing) flow may still make the QoS estimation very dependent on the traffic fluctuations and therefore reduce the performance. Moreover, the probing traffic in the network can be considerable because a probing flow is sent for each new flow. Thrashing may occur in out-of-band probing [bres00b], since even though the number of accepted flows is small, simultaneous probing by too many flows can lead to false rejections (in the case of in-band probing, this situation can lead to a collapse, since accepted flows would lose their QoS guarantees). Finally these schemes also exhibit unfairness for multirate and multihop flows [bres00b].

3.5.2 Active measurement-based edge-to-edge schemes with per-aggregate probing

In these schemes, a single probing flow, related to an aggregation of accepted flows, is generated and sent continuously through the path and its QoS is measured in order to be used in the AC decision. The procedure is the following:

- For an aggregation of accepted flows in a path, a single probing flow travels continuously from the ingress to the egress through the path.
- The egress node continuously measures the QoS experienced by the probing flow (i.e., in each particular time interval).
- The egress node reports the measured QoS through a signaling packet to the ingress node (periodically, when a defined threshold is exceeded, etc.), which performs the AC decision, or alternatively, it decides and then informs the ingress node.
- According to the AC decision, a new flow is allowed to transmit or not. The AC decision is fast as it is made as soon as the new flow arrives.

One example of these schemes is [lima04a], in which it is used to build an edge-to-edge AC for a multiservice network based on packet classes [lima03a]. A probing flow, which contains timestamping and sequencing information, is sent for each packet class and path, and multiple QoS parameters are measured at the egress, such as delay, jitter and loss, although this depends on the packet class characteristics. Each measured QoS parameter (updated in each time interval) is then compared to a threshold to decide whether new flows can be admitted or rejected. The threshold is based on a target value of the packet class and also on a safety margin. This is necessary since the AC algorithms do not use the traffic parameters of the new flow. Probing is in-band but with a low rate (to not interfere with the data traffic), and uses a pattern that has been especially designed to better capture the QoS parameters of packet classes. In comparison with per-flow probing this scheme avoids per-flow intrusive probing traffic; the AC is fast, made at once since measurements are available online; and it increases the confidence level of measurements since they are achieved using many samples instead of just a few. However, the AC algorithms do not immediately consider the effect of recently accepted flows until future measurements take them into account, but this takes some time. Therefore, a high rate of new arriving flows to an ingress-egress pair may cause false acceptances. The same problem holds for concurrently accepted flows in other ingress nodes. Some proposed solutions, which may lead to lower utilization, are increasing safety margins in the AC algorithms, or using some degree of over-provisioning, or a rate-based credit system controlled by egress nodes [lima06a].

3.5.3 Passive measurement-based edge-to-edge schemes

In these schemes, the QoS of the aggregation of accepted flows is continuously measured at the egress to be used in the AC decision. No probing flow is generated. The procedure is the following:

- The egress node continuously measures the QoS experienced by the aggregation of accepted flows in a path (i.e., in each certain time interval).
- The egress node reports the measured QoS through a signaling packet to the ingress node (periodically, when a defined threshold is exceeded, etc.), which performs the AC decision, or alternatively, it decides and then informs the ingress node.
- According to the AC decision, a new flow is allowed to transmit or not. The AC decision is fast as it is made as soon as the new flow arrives.

One example of these schemes is [ceti01a], in which it is used to build a service with statistical guarantees on the maximum delay. It uses the theory of traffic envelopes to describe the aggregated rate of traffic at the ingress ("arrival envelope"), the available service in the path ("service envelope") and the traffic characteristics of the new flow ("flow envelope"). Packet's arrival times at the ingress (which are used to obtain the arrival envelope) are written on packets in order to measure their delay at the egress (which is used to obtain the service envelope). Besides continuously measuring, the egress node makes the AC decision. The new flow specifies the required service to the ingress node through RSVP packets, which are forwarded to the egress node. The AC algorithm considers the maximum new flow envelope (e.g., the peak rate), the desired delay bound, the desired maximum violation probability, the mean and the variance of the measured maximum arrival envelope, and the mean and the variance of the measured minimum service envelope. Besides inserting timestamping information, sequencing and ingress identification is required.

Another example is [zhu06a], in which it is used to build a service with statistical guarantees on the loss rate. This scheme uses the concept of "achievable" capacity of the path, i.e., an equivalent capacity of the path, and assumes that the aggregated traffic rate follows a Gaussian distribution. With this model it is easy to relate the aggregated traffic rate at the ingress, the "achievable" capacity and the loss rate in the path. Then the actual loss rate in the path is measured at the egress by counting the lost packets in a defined time interval (packets carry a sequence number). This information is periodically reported to the corresponding ingress node. The aggregated rate of accepted traffic is measured periodically at the ingress (characterized with its mean and variance), and, together with the actual loss rate, is used to obtain the actual "achievable" capacity. The mean and variance of the new flow's rate, the mean and variance of the actual aggregated rate, and the actual "achievable" capacity are used to estimate the future loss rate if the flow has been accepted, which is compared to the target loss rate to make the AC decision.

The main advantage of these schemes compared to hop-by-hop AC schemes is the edge-to-edge architecture. They do not require changes in the network core, where reservation state is not maintained (neither per-flow nor per-aggregate), and only edge nodes are AC aware. In comparison with active measurement-based edge-to-edge AC schemes with per-flow probing, these schemes avoid any intrusive probing traffic, the AC is fast as it is made at once since measurements are available online, and the confidence level of measurements is increased since they use many samples instead of just a few. However, packets are required to carry information such as sequencing, timing, or ingress identification. Moreover, the AC algorithms do not immediately consider the effect of a recently accepted flow until future measurements take it into account, but this takes some time. Therefore, a high rate of new arriving flows to an ingress-egress pair may cause false acceptances. The same problem holds for concurrently accepted flows in other ingress nodes.

3.6 Conclusions

In this chapter we have studied the main AC schemes that have been proposed in the Internet, focusing on the simplicity of their architectures in terms of the number of nodes that participate in the AC, the required state, the use of signaling, and others.

The majority of AC schemes we have studied deal with real-time traffic and only a few with elastic traffic (and very few for both traffic types). We have classified them mainly according to the nodes that participate in the AC decisions, i.e., centralized schemes, hop-by-hop schemes, one-hop schemes on LPs with reservation and edge-to-edge schemes:

- In centralized schemes there is a single AC entity in the network (where the AC algorithm is implemented) that maintains the state of resource usage and exchanges signaling packets with the ingress nodes when new flows arrive. Flow's reservations can be based on state or on occupancy. The network remains simple since the state is not distributed. Service requests are processed serially and unlike distributed schemes, they do not have the problem of simultaneous AC decisions. However, in large and highly dynamic networks the centralized entity would have to process too many signaling packets and might become a traffic bottleneck. Moreover, a centralized approach is highly vulnerable to failures.
- In hop-by-hop AC schemes, each node maintains the actual aggregated reservation and performs a local AC decision through a link-scope AC algorithm based on measurements or parameters. Flows' reservations can be based on state or on occupancy. Usually the duration of the AC phase (and also the time an accepted flow has to wait for starting to transmit) is about RTT. As they are distributed, these schemes are more robust and able to cope with highly dynamic networks. However, all nodes are aware of AC. Moreover, some of these schemes maintain the per-flow state in nodes and many often use per-flow signaling packets to communicate between nodes (although some schemes use a lightweight signaling or even no signaling).
- In one-hop AC schemes on LPs with reservation only the ingress node is involved and maintains its flows' reservations based on state or on occupancy (although the previous reservation in the establishment of the LP can involve either all the nodes of the path or a centralized entity and is based on state). The AC decision is simple, only at the ingress node and without per-flow signaling. It is also fast, as it is made at once. However, per-path resource reservations require more complex management to avoid that the LPs that pass through the same link share resources in a non-appropriate way and consequently false rejections occur.
- In edge-to-edge AC schemes only the ingress and egress nodes participate in the AC decision. There is an edge-to-edge scope AC algorithm implemented at the ingress or egress node, which is based on measurements performed at the egress and sent back to the ingress through special packets (usually signaling packets). Flows' reservations are based on occupancy. These schemes do not require changes in the core, since the nodes of the path do not maintain either a per-flow or aggregate reservation state (they are not aware of AC) and do not exchange signaling packets. However, some of them generate intrusive (signaling) traffic that can be considerable and the duration of the AC phase (and also the time an accepted flow waits before transmitting) may be quite long. In others the AC algorithms do not consider the effect of a recently accepted flow until future measurements take it into account. Since this takes some time, a high rate of new arriving flows to an ingress-egress pair can cause false acceptances. Other schemes require that data packets carry information such as sequencing, timing information, or ingress identification.

Using implicit ways of communication between nodes and traffic measurements in the AC algorithms can considerably reduce the number of nodes that are aware of AC, the state they maintain and the signaling required. The implicit way (e.g., through the port numbers in data packets, marks in other packet fields, predefined values of parameters, etc.) has the advantage

of not requiring signaling packets. Using active or passive measurements results in a flow's reservations being based on occupancy. They do not require a per-flow state, are maintained as long as the flow transmits (following a certain traffic profile) and are released when the flow is inactive, without needing signaling packets. Moreover, measurements may increase resource utilization, although they run the risk that traffic fluctuations may cause eventual QoS degradations; therefore, they provide qualitative guarantees. Also note that although per-flow state is not required, in some other nodes (e.g., in the ingress node of the path, in a centralized entity, etc.) a list of the accepted flows should be maintained to differentiate between the packets of new flows from the ones of accepted flows, and also to be able to isolate (or protect) the accepted flows by enforcing the agreed traffic profile on their input traffic.

Chapter 4:

Network schemes for TCP elastic traffic proposed in the Internet

In this chapter we review the main network schemes that have been proposed in the Internet for TCP elastic traffic. As we discussed in Section 2.4, this traffic requires a minimum throughput from the network and if possible, an extra throughput, a service we have called MTS. We describe these network schemes classified into two broad groups: the ones that do not use AC and the ones that do use it. For each of these we describe the main characteristics of the service provided and its architecture.

4.1 Network schemes for TCP elastic traffic without AC

In this section we review the main schemes that have been proposed in the Internet to provide a network service for TCP elastic traffic, when the mechanisms used are basically traffic conditioning and/or queue disciplines, and AC is not considered. In consequence (see Subsection 2.2.3), when resources in the followed network path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied; otherwise, i.e., during congestion situations, none of them is satisfied. We say that the network service has a relative guarantee (see Subsection 2.1.4), since the throughput received by a flow is defined as a function of the throughput received by other flows. For example, in a fair throughput service, the goal is to provide a throughput equal to the fair rate of the bottleneck link, i.e., the link's capacity divided by the number of present flows (in fact, the max-min fairness, see Subsection 2.4.3); or in the weighted version, the proportional throughput service, flows' throughputs and flows' weights are proportional, and therefore different throughputs can be provided. Congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques. If the network is over-provisioned so that congestion never or rarely occurs, these schemes always provide a service with absolute guarantees.

A possible scheme would be using FQ or WFQ scheduling (see Subsection 2.2.2) for all flows in all routers. With FQ each flow would receive the max-min fair rate, while with WFQ they would receive the weighted max-min fair rate (and therefore throughput differentiation according to flows' weights). Isolation between flows would be provided by queues without

needing specific traffic conditioning mechanisms. However, this scheme would be too complex because it would require per-flow state and per-flow management in all routers. For each arriving packet, the router would need to classify the packet into a flow, update some per-flow variables and perform per-flow operations. Per-flow state should be established and updated explicitly through per-flow signaling (although this would result in a high overhead given that most elastic flows are short, see subsection 2.4.4), or implicitly through flows' data packets and timeout procedures.

We review the following set of schemes for TCP elastic traffic without AC in the next subsections. Firstly, the "traditional" scheme in the Internet with FIFO and Tail Drop queues, then a set of schemes based on packet classes, the scheme based on Core-stateless Fair Queuing and finally the User-Share Differentiation scheme. For each of these, we describe the main characteristics of the service, that is, whether they provide the same or different throughputs, and whether they provide isolation between flows (so that flows sending more traffic than their allocated throughput do not damage well-behaved flows that do send according to their allocated throughput), and also the architecture of the scheme, that is, the specific mechanisms used, the required state and the use of signaling.

4.1.1 The "traditional" scheme

The "traditional" scheme in the Internet is based only on FIFO and Tail Drop queues. Traffic conditioning mechanisms are not used. All packets receive the same treatment and the service provided is best-effort (see Subsection 2.2.3). The main advantage of the scheme is the simplicity. However, it does not provide any isolation between flows, and in the case of traffic overload, flows injecting more traffic "steal" resources from the rest (the output average rate is proportional to the input average rate).

The combination of this scheme and TCP rate-adaptive algorithms (see Subsection 2.4.3) results in a fair throughput service. The goal is to provide a throughput equal to the fair rate of the bottleneck link, i.e., the link's capacity divided by the number of present flows, although the effective resource sharing may exhibit unfairness in some situations (flows with large RTT versus flows with small RTT, or short flows versus long flows). An obvious consequence is that it is not possible to provide different throughputs to different TCP flows. Moreover, the fair throughput service is achieved by TCP sources through a probing method, increasing the sending rate while there is no congestion (e.g., no packet loss) and decreasing it when congestion occurs, oscillating around the fair rate. Therefore, this approach relies on cooperation between sources that implement the same algorithms. An advantage is that no support from the network is needed, since the fair rate is not indicated to the TCP sources nor enforced. However, some sources may not react against congestion (e.g., real-time sources that do not decrease the sending rate) or react in a different way, so that well-behaved TCP sources may receive smaller throughput (i.e., they are not protected).

An enhancement of the "traditional" scheme is achieved by replacing Tail Drop by an Active Queue Management such as RED [floy93a, rfc2309]. One of the main goals of RED is to avoid the so-called "TCP global synchronization problem", which arises from the interaction between TCP rate-adaptive algorithms and Tail Drop, in the following way: when a sequence of packets arrives and the queue occupancy is high, multiple packets may be discarded; flows experiencing this packet loss will decrease the sending rate at a similar time, and after a while, when losses do not occur, they will increase the sending rate at a similar time, and so on, becoming "synchronized". Moreover, it is likely that the number of losses in a single flight of packets of a flow is large, resulting in RTO expiration (a severe congestion indication), the flow entering Slow Start, and a strong reduction in the sending rate. The synchronized

behavior together with the burstiness of TCP traffic leads to poor link utilization and low aggregated throughput. RED works in the following way:

– It measures the queue's average occupancy, *avg*, by using a low-pass filter or exponentially weighted moving average of the instantaneous queue occupancy.
– It discards packets before the queue is full according to a dropping probability P_{drop} that depends on the average occupancy *avg* and two thresholds, *min* and *max* (see Figure 4-1): when *avg* < *min*, no packet is dropped; when *max* < *avg* < *min*, the packet dropping probability increases linearly with *avg*, from probability 0 to *Pmax*; when *avg* > *max*, all packets are dropped. Therefore, the dropping probability of the arriving packet is higher as *avg* increases.

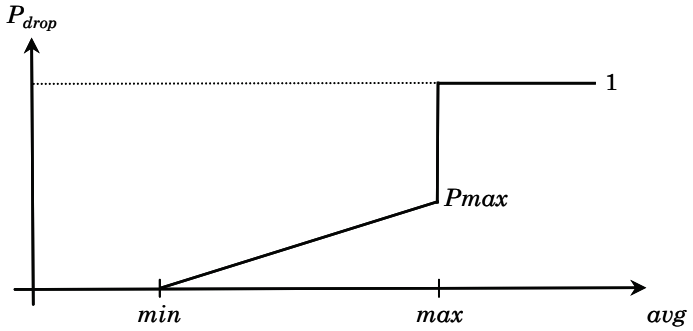


FIGURE 4-1: Dropping probability as a function of the queue's average occupancy in RED.

Since RED uses an average occupancy, short bursts of packets (sort-term congestion) are filtered, and thus ignored, without inducing packet loss. However, when bursts are longer (long-term congestion), the average occupancy increases and packets start to be discarded to indicate the congestion to sources. Note that RED detects incipient and light congestion and provides an early indication: before several packets are discarded (indication of severe congestion), it is likely that a single packet in a flight of packets of a flow is discarded, resulting in Fast Retransmit, the flow entering Congestion Avoidance, and a weak reduction of the sending rate. The probabilistic discarding avoids global synchronization, since only some of the sources will experience packet loss and decrease the sending rate. When congestion is stronger, the indication to sources is much more frequent (P_{drop} increases with *avg*). Moreover, the probabilistic packet discarding will tend to affect flows causing the congestion more (the ones that receive higher throughput), since most of the arriving packets will belong to them. Finally, note that when RED is used together with ECN [rfc3168], packets are marked instead of being discarded, on the assumption that sources will react in the same way as if a packet were lost. Numerous references about RED can be found in Sally Floyd's homepage [floyd].

4.1.2 Schemes based on packet classes

These schemes are based on packet classes in a similar way to the Diffserv architecture (see Subsection 2.3.3). The mechanisms used are the following:

– Traffic conditioning mechanisms at the network ingress assign each flow's packet to a class and write a mark in the packet's header that identifies the class (the number of classes is small), according to an agreed traffic profile. Alternatively, the packets may arrive at the network ingress already marked (e.g., previously by sources), and then traffic conditioning at the network ingress checks and enforces the agreed traffic profile (out-profile packets can be remarked or even discarded).

– Queue disciplines in the network core are based on classes, i.e., they apply a different treatment to packets belonging to different classes.

Per-flow state is only kept at the edge while the core remains simple and highly scalable. Using traffic conditioning and class-based queues can allow isolation between flows and different throughput to different flows to be provided, as well as the possibility to coexist with other different network services.

The *in* and *out* scheme of the Assured Service

The Assured Service, defined within the so-called "allocated-capacity" framework in [clar98a], is able to provide different throughputs to flows from different users during congestion. Moreover, it protects TCP flows against non-responsive sources. The proposed scheme uses two packet classes, called *in* and *out*, with different discarding priorities:

– There is an input traffic profile (for each user) that defines the flow's desired minimum throughput r_{min}. The average sending rate of the flow r is measured and compared with r_{min} in order to classify each packet as an in-profile or out-profile. The goal of this classification is to obtain a sequence of in-profile packets with a rate equal to the minimum throughput, specifically, $min(r, r_{min})$, and a sequence of out-profile packets with a rate equal to the exceeding traffic above it, i.e., $min(0, r\text{-}r_{min})$. Packets are marked accordingly as *in* or *out*.
– There is a single FIFO queue (to maintain packet ordering) with priority discarding so that if a packet has to be discarded, the *out* class has a higher discarding priority than the *in* class (this behavior was generalized and standardized by the Diffserv Working Group in the definition of the AF PHB [rfc2597]).

As a consequence, *in* packets have a higher assurance of delivery than *out* packets. The desired minimum throughput is provided when the aggregated *in* traffic does not cause an overload in any of the links of the network path. When an overload occurs, the throughput provided to each flow is a share of the bottleneck link's capacity that is proportional to (and smaller than) the desired one. Therefore, the difference between the provided throughputs during congestion comes from the different desired throughputs of the input traffic profile of flows (users).

The following algorithms were proposed in [clar98a] to implement this scheme (similar algorithms were also proposed in [feng99a]):

– The flow's average rate is estimated using the TSW algorithm, and the marker is based on a probabilistic function.
– The priority discarding in queues uses the RIO algorithm.

TSW provides a smooth estimate of the TCP sending rate in a way suitable to the burstiness of TCP traffic. The average sending rate *avg_rate* is estimated upon each packet arrival and over the last period of time (or window), which considers a "past history" equal to the so-called *win_lenght* parameter. The algorithm is simple since the only state variables are the arrival time of the previous packet and the previous value of *avg_rate*. A difficulty is that the recommended value for *win_lenght* is the flow's RTT, which is usually not known at the network ingress. Therefore, a fixed value has to be used and the average rate is not optimally estimated. A proposed solution is to implement this algorithm and the marking in the TCP source itself, which has an estimate of the actual RTT (in this case, the network would then check and enforce the agreed traffic profile).

The marker is based on a probabilistic function. Once *avg_rate* for the arriving packet is calculated, the marker decides whether the packet is *in* or *out* in the following way: if *avg_rate* is smaller than the desired throughput R_T, the packet is *in*; otherwise, the packet is *out* with probability $P_o = (avg\_rate - R_T)/avg\_rate$ or *in* with probability $1-P_o$ (a variant is using $1.33 \cdot R_T$

instead of R_T as a threshold). The probabilistic function is used to space *out* packets and to reduce the probability of consecutive drops in a single flight of packets, which could lead TCP to enter Slow Start and severely reduce the sending rate. The design of TCP markers has been a subject of research and there have been more proposals (e.g., [feng99a, kuma02a]).

The RIO algorithm extends RED to work with two classes. Two sets of parameters are used and two separate average buffer occupancy calculations are tracked, one only for *in* packets and another one for all (*in* plus *out*) the packets (see Figure 4-2):

– The dropping probability of *in* packets depends only on the buffer occupancy of *in* packets avg_{in}, with parameters min_{in}, max_{in}, $Pmax_{in}$.
– The dropping probability of *out* packets depends on the buffer occupancy of *in* plus *out* packets avg_{tot}, with parameters min_{tot}, max_{tot}, $Pmax_{out}$.

RIO's objective is to discriminate *out* packets from *in* packets: when there is incipient congestion, RIO first drops some *out* packets; if the congestion persists, RIO drops all the *out* packets; finally, *in* packets are only dropped when the router is flooded with *in* packets. Therefore, RIO parameters have to be chosen carefully (e.g., [40, 70, 0.02] for *in* and [10, 30, 0.2] for *out*, are one of the choices in [clar98a]).

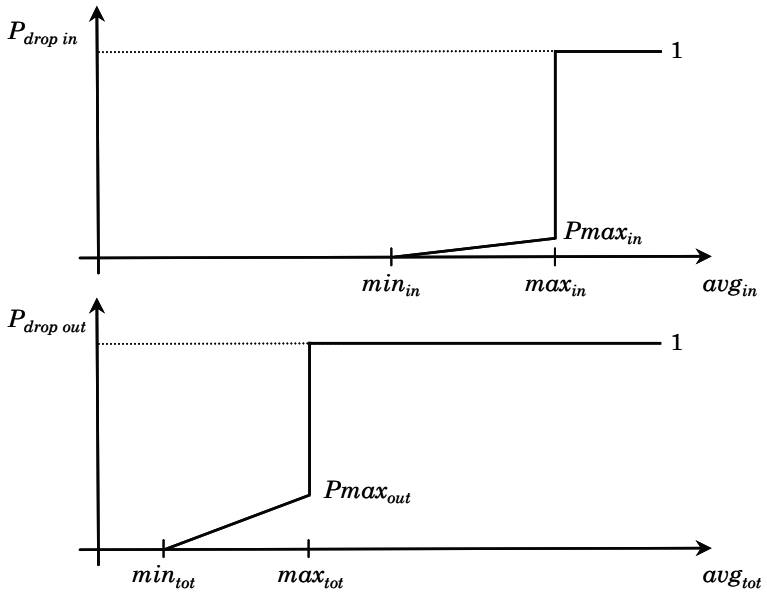


FIGURE 4-2: Dropping probability as a function of the queue's average occupancy in RIO.

TCP-state based differentiation

This scheme [nour02a] uses three packet classes, called *high*, *med* and *low*, with different discarding priorities (following the AF PHB definition [rfc2597]):

– Packets are marked in the host by TCP sources as *high*, *med or low*. The marking algorithm depends on the TCP state, i.e., on the actual value of the window and on the identification of some "important" packets, as we describe below in more detail.
– An agreed traffic profile specifies the total aggregated rate of *high* and *med* packets per user, in the form of two token bucket profiles. At the network ingress, traffic conditioning mechanisms enforce this traffic profile, and out-profile packets can be remarked to a lower priority or even discarded. Note that traffic conditioning is made over the user's flow aggregate, and therefore it is in the best interest of sources to mark packets in conformance with the agreed traffic profiles.

– There is a single FIFO queue (to maintain packet ordering) with priority discarding so that if a packet has to be discarded, the *low* class has the highest discarding priority, the *med* class the medium discarding priority, and the *high* class the lowest discarding priority. The algorithm in queues is an extension of RED for three classes (like RIO is an extension of RED for two classes).

The marking algorithm at the TCP source considers two cases:

– In the first case, the marking is based on the actual value of the window (since the average sending rate – in RTT – is roughly equal to the window size divided by RTT). The algorithm considers that if a connection is performing well and the window is high, there is no need to protect its packets and it is better to use the *high* marks to improve the performance of other connections that need it; if then the connection suffers packet drops and its window is reduced, marking its packets as *high* can help it to recover. Following these ideas, the window-based marking compares the actual window *cwnd* with two thresholds, $high_{thresh}$ and med_{thresh}, in the following way: if $cwnd \leq high_{thresh}$, packets are marked as *high*, if $high_{thresh} < cwnd \leq med_{thresh}$, packets are marked as *med*, and if $cwnd > med_{thresh}$, packets are marked as *low*.
– In the second case, the algorithm identifies some "special" packets, the ones that are more important for the stability of the TCP congestion control algorithms, and marks them as *high*. Specifically, these packets are the connection establishment packets (important for the initial RTT measurement and RTO calculation), the data packets sent when the window is small (since TCP is more vulnerable to losses), and the data packets retransmitted after an RTO expiration or Fast Retransmit (since their loss could lead to RTO expiration). Note that some of these packets could also be marked as *high* by the window-based marking.

This scheme compensates for the unfairness experienced by short TCP flows. As we have seen in subsection 2.4.3, short flows tend to achieve smaller throughput than long flows because their initial window is usually small and because they are more vulnerable to losses. This scheme identifies these situations and prioritizes packets to reduce losses; therefore, it tends to provide a fair throughput service, i.e., to share network resources equally between flows. Moreover, note that traffic conditioning mechanisms at the network ingress provides isolation between flows from different users.

Throughput differentiation is achieved by using different marking thresholds for different flows. The differentiation can be at the application level (e.g., higher thresholds for web than for ftp), at the user level (e.g., demanding users have higher thresholds than "normal" users), or even for different transferred documents (e.g., higher thresholds for transferring basic web files than for the rest of the web page objects).

Preferential treatment to short TCP flows

These schemes [guo01a, avra04a] give preferential treatment to short flows over long flows by using different packet classes. The aim is two-fold: to compensate for the unfairness experienced by short flows, which tend to get less than their fair share when they compete for the bottleneck link's capacity; and giving priority to short flows, which has been shown to reduce the transfer time of short documents without hurting the performance for long flows, when considering heavy tailed document size distributions (see Subsection 2.4.3).

Neither scheme provides isolation between flows. Two packet classes are used, e.g., called *short* and *long*:

– At the network ingress, the first packets of each flow are marked as *short* and the rest as *long*, according to a defined threshold.
– *Short* packets are preferentially treated over *long* packets in queues.

Note that the proposed *short* and *long* marking does not result in a classification between short and long flows, since the first packets of long flows are also marked as *short*. However, this is a desired feature, since in fact the unfairness is not between short and long flows but rather between the first packets of flows and the rest of the packets (i.e., the first packets of long flows experience the same problems). Therefore, the preferential treatment to the *short* class helps all flows.

The preferential treatment in queues in [guo01a] is based on RIO, i.e., *short* packets are discarded less than *long* packets, so that they experience fewer losses. The preferential treatment in [avra04a] is based on priority queuing, i.e., *short* packets are served before *long* packets, so that they experience fewer losses and smaller delays (also note that packet ordering in a flow is still maintained).

4.1.3 Core-Stateless Fair Queuing (CSFQ) based scheme

This scheme [stoi03a] provides a fair throughput service as well as isolation between flows. Moreover, by assigning a weight to the flow, it can be extended to provide different throughputs to different flows, proportionally to the flows' weights.

The scheme uses the CSFQ algorithm in queues, which closely emulates the behavior of the FQ algorithm (see Subsection 2.2.2), but without needing a per-flow state. Instead, per-flow state is carried by packets using the DPS technique (see Subsection 3.3.3): the state variables are encoded in the packet's header and then are used and modified by the queue disciplines. In this scheme the state is the flow's rate:

- The incoming rate of each flow is estimated at the network ingress and a label is written on its packets carrying the value of this rate.
- The queue discipline uses FIFO together with the CSFQ algorithm, which probabilistically discards packets so that each flow receives the fair rate in the link. CSFQ only uses the packet's label and measurements over aggregates.

The ingress router, upon each packet arrival, classifies the packet into a flow, updates the estimation of the flow's rate r, and labels the packet with r. This estimation is based on an exponential weighted moving average of the instantaneous rate (with a weight that depends on the packet inter-arrival time).

CSFQ in all routers works as follows:

- Each router periodically estimates the fair rate f in the link.
- Upon receiving a packet labeled with incoming rate r, the router drops the packet with probability $P_{drop} = max[0, (r-f)/r]$. Therefore, if $r \leq f$, all packets of the flow are forwarded and the flow's output rate is kept to r; if $r > f$, some packets of the flow are probabilistically discarded (hopefully, $(r-f)/r$ is the fraction of discarded packets), so that the output rate is approximately decreased to f. In any case, when a packet is forwarded, the router updates the packet's label with the flow's output rate (the minimum between f and the incoming r), which is the new flow's arrival rate for the next router.

The fair rate f in the link is estimated at certain times. The router continuously measures the aggregated incoming rate A and the aggregated forwarded rate F (both with the same procedure used for the flow's incoming rate at the ingress) in the link of capacity C. If there is no congestion ($A < C$), f is chosen as the maximum flow's rate between the flows that traverse the link, i.e., the maximum packet label observed in that time (and therefore the discarding probability is 0 for all packets of all flows). If there is congestion ($A \geq C$), a heuristic and iterative algorithm varies f (and therefore P_{drop} and F) by a factor C/F until it converges, i.e., until F matches C.

This scheme provides a fair throughput service and isolation between flows without needing a per-flow state in the core. However, it requires the state to be processed and updated for each packet in each router, and the state in the packet's header to be encoded.

4.1.4 User-Share Differentiation (USD) scheme

This scheme [wang00a, wang97a] is able to provide different throughputs to flows from different users proportionally to some agreed users' weights, but in an aggregated way. Moreover, it provides isolation between flows from different users. The basic points of the USD scheme are the following:

- Each user has a weight (defined in a user-provider agreement), which controls resource sharing between users for both its sending and its receiving traffic.
- The queue discipline uses the WFQ algorithm (see Subsection 2.2.2) or similar, which shares the link's capacity fairly between the traffic from different users according to the weights.

The user is chosen as the basic unit that defines traffic control granularity, so that all traffic that has originated from a user or destined to a user is aggregated within the network (within the traffic of a single user it is up to the user to decide how the service is shared internally). The state needed inside the network is reduced since it is not flow-based but rather user based. In each router there is a table with the user identifier and its associated weight (the user identifier can be the IP address of an end-user, the network prefix for a network, or a set of network prefixes for a group of networks). The per-user state makes the scheme highly scalable in the hierarchical structure of recursive user-provider relationships of the Internet.

The user identifier and its corresponding weight could be distributed to routers inside the network through a network management protocol. For each arriving packet, the router looks up the weight of the sending user and the weight of the receiving user in the table, since both weights control the sharing. This conflict is solved by making the WFQ scheduler use the minimum of the two weights.

Isolation between traffic of active users is provided by WFQ without needing specific traffic conditioning mechanisms at the network ingress. If one user transmits more than its actual allocated throughput in a given link, it will cause its own packets to be dropped in the queues.

4.2 Network schemes for TCP elastic traffic with AC

In this section we review the main schemes that have been proposed in the Internet to provide a network service for TCP elastic traffic when the mechanisms used are basically traffic conditioning, queue disciplines and AC. Therefore (see Subsection 2.2.3), when resources in the followed network path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied; otherwise, i.e., during congestion situations, some of them receive the minimum throughput with absolute guarantees (they are "accepted") and the rest do not receive it (they are "rejected" or "blocked"). Congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques. If nevertheless, congestion occurs, using AC achieves an efficient use of network resources by maximizing the number of satisfied flows, although it complicates the network scheme. The blocking rate depends on the behavior of users' demands, the chosen network provisioning, the routing techniques used and the ability of the AC mechanism to maximize the number of satisfied flows.

A possible scheme would be using FQ or WFQ scheduling (see Subsection 2.2.2) for all flows in all routers and a classical parameter-based hop-by-hop AC (see Subsection 3.3.1). With FQ each flow would receive the same minimum throughput and an extra throughput equal to the

max-min fair share of the remaining resources. With WFQ different flows would receive different minimum throughputs according to the assigned weights, and an extra throughput equal to the weighted max-min fair share of remaining resources. Isolation between flows would be provided by queues without needing specific traffic conditioning mechanisms. Per-flow signaling would carry the flow's minimum throughput request from router to router through the path, and each router would perform a local AC decision to limit the number of flows in each link so that accepted flows would receive their desired minimum throughput. However, this scheme would be too complex. It would require per-flow state and per-flow management in all routers. Given that most elastic flows are short (Subsection 2.4.4), using per-flow signaling would imply a high overhead and a rather long duration of the AC phase.

We review the following set of schemes for TCP elastic traffic with AC in the next subsections. Firstly, the scheme for a guaranteed throughput service in the Corelite architecture, then the implicit AC for TCP connections and finally the scheme for elastic traffic in the Cross-Protect architecture. For each of these, we describe the main characteristics of the service, that is, whether the minimum throughput can be different or is the same for all flows, the expected extra throughput that results from sharing the remaining resources, and whether isolation between flows is provided (so that flows sending more traffic than their allocated throughput do not damage well-behaved flows that do send according to their allocated throughput), and also the architecture of the scheme, that is, the specific mechanisms used, the required state and the use of signaling.

4.2.1 The scheme for a throughput service in Corelite

The Corelite architecture provides several throughput and delay services using the same set of basic mechanisms. The scheme for a throughput service [siva00c, siva99b] is able to provide different minimum throughputs r_{min} to different flows, and an extra throughput according to a weight w. It has two modes, which differ in the kind of guarantees: it is deterministic in the "guaranteed" mode (there is no loss if the sending rate is not higher than r_{min}), and it is qualitative in the "predictive" mode (low loss if the sending rate is not higher than r_{min}). It provides isolation between flows through traffic conditioning at the network ingress. Per-flow signaling is used to indicate the start of the flow, the requested r_{min} and the AC response. Per-flow state in the core is not required. We explain the AC scheme below, but firstly we describe the mechanisms used when a flow (in either mode) has already been accepted by AC:

- The ingress router performs traffic conditioning over the flow depending on the comparison between the actual flow's average rate r and two thresholds, r_{min} and r_{max}, where r_{min} is the minimum throughput and r_{max} is equal to r_{min} plus an extra throughput that is adapted according to the feedback received from core routers. A token bucket algorithm is used for measuring r. Flow's packets are classified into three types (resulting in three "subflows"): in-profile packets, with a rate equal to $min(r, r_{min})$; out-profile packets, with a rate equal to $min(0, r\text{-}r_{min}, r_{max}\text{-}r_{min})$; and the exceeding packets, with a rate equal to $min(0, r\text{-}r_{max})$. The exceeding packets are discarded; the other packets are forwarded and some of them may also be turned into special packets called "markers", as we explain in the next point.
- The ingress router periodically turns some flow's packets into markers: one marker is inserted for every N number of "data" packets (or bytes) and each marker carries N. Therefore, the transmission rate of markers taking into account the carried N reflects the rate of the flow. Markers carry the source address of the ingress router, a unique identification of the flow within the ingress router, and the number of "data" packets (or bytes) that they represent. Markers are logically distinct packets, but are physically piggybacked to a "data" packet. The use of markers differs if the mode is "predictive" or "guaranteed", as we explain in the next point.

– For "predictive" flows, *p*-markers for in-profile packets and *w*-markers for out-profile are used. One *p*-marker is introduced for every $N_p = K_1 \cdot r_{min}$ "data" packets (or bytes) of in-profile traffic, where K_1 is a constant. Each *p*-marker carries N_p. One *w*-marker is introduced for every $N_w = K_2 \cdot w$ "data" packets (or bytes) of out-profile traffic, where K_2 is a constant and *w* is the weight. Each *w*-marker carries N_w. If the actual flow's rate is smaller than r_{min}, the rate of *p*-markers reflects this rate (and no *w*-markers are introduced); if the actual flow's rate is greater than r_{min}, the rate of *p*-markers reflects r_{min} and the rate of *w*-markers reflects the extra rate of the flow (above r_{min}), normalized according to the weight *w*.
– For "guaranteed" flows, *g*-markers for in-profile packets and *w*-markers for out-profile are used, in a similar way as for "predictive". The only difference is that if the actual flow's rate is smaller than r_{min}, the rate of *g*-markers does not reflect this rate but rather the minimum r_{min}.
– Routers use FIFO queues. They also extract the markers from packets and maintain a queue of *p*-markers, *g*-markers and *w*-markers. When congestion is detected, a random number of *w*-markers are selected and sent back to the ingress router that generated it (if there were no *w*-markers, firstly *p*-markers and then *g*-markers would be selected, but this is not likely to happen as AC is used). Also note that the *w*-markers of flows with a greater weight *w* are less likely to be selected.
– Periodically, the ingress router checks the markers received from core routers during the last time period corresponding to a flow. The flow's threshold r_{max} is reduced in proportion to the received markers, and if no marker has been received, it is increased additively by a constant.

The AC scheme is hop-by-hop since each router performs a local AC decision. Per-flow signaling carries the AC requests and responses, but it does not require a per-flow state in the core. The duration of the AC phase is about one RTT. The scheme is the following:

– A request signaling packet with the rate requirement r_{min} is sent along the path.
– Each router maintains the available bandwidth B_{av}, which is calculated using the received markers and updated at a certain time period.
– A router in the path receives the request packet. If the request can be accepted ($r_{min} < B_{av}$), the router reduces B_{av} by r_{min} and forwards the request packet to the next router; otherwise a reject response signaling packet is sent back to the ingress.

At the beginning of a given time period, each router knows the available bandwidth B_{av} for this period. A request is accepted if r_{min} is available. If so, B_{av} is reduced by r_{min} and the resulting value of B_{av} is used for the next AC decision until the end of this time period. During this time period the router estimates the value of B_{av} to be used for the next time period. It is calculated from the number of *p*-markers and *g*-markers received over that time (the router counts the number of packets – or bytes – that the marker represents, which is carried by the marker). Note that in B_{av}, due to the different ways that *p*-markers and *g*-markers are generated, the aggregated rate of "predictive" flows is based on "real" measurements that aim to take into account the multiplexing gain, while the aggregated rate of "guaranteed" flows is based on "virtual" measurements (see Subsection 3.3.3) that aim to equal the sum of the reserved rates of flows (i.e., based on their declared traffic parameters). Therefore, the "guaranteed" mode provides a deterministic guarantee, probably at the cost of reducing resource utilization, while the "predictive" mode can be more efficient in using resources but it provides a qualitative guarantee.

This scheme is able to provide different minimum throughput to different flows and isolation. It does not require per-flow state in the core or per-flow queuing. However, it requires per-flow signaling, which could result in a high overhead, a rather long duration of the AC phase, and quite complex management for the markers. Finally, there are no details about how TCP flows are defined and identified in the scheme.

4.2.2 Implicit AC for TCP connections

These schemes [kuma00a, mort00a] provide the same minimum throughput to all flows, which here are defined as TCP connections. The guarantee is qualitative. It does not provide isolation to accepted flows. The start of the flow (connection) and the AC response are implicitly indicated without signaling. The AC is fast. Per-flow (connection) state is not required in any router. Therefore, their main advantage is the simplicity. The mechanisms used are the following:

- The queue discipline is FIFO and there is no traffic conditioning mechanisms at the ingress, as in the "traditional" scheme (see Subsection 4.1.1). Therefore, isolation between flows is not provided and the fairness in resource sharing between the accepted flows depends on the TCP rate-adaptive algorithms.
- The AC scheme is based on measurements and without signaling. The AC algorithm measures the actual use of resources through a particular parameter, which is compared to a threshold to make the AC decision.

The authors only consider the AC in a single link, although it could be extended to a hop-by-hop scheme (or obviously to a one-hop scheme on LPs with reservation). The start of the flow (connection) is implicitly indicated to the router through its first packet, i.e., the TCP connection establishment packets (SYN or SYN/ACK). The AC response is also implicitly indicated to the flow: in the case of acceptance, the connection establishment is allowed to proceed by forwarding the detected establishment packet; in the case of rejection, the connection establishment is aborted, by sending an RST packet to the sender [kuma00a] or by discarding the detected establishment packet [mort00a]. Therefore, the AC is simple and moreover, it is fast as it is made as soon as a new flow arrives (in a hop-by-hop scheme, no signaling packet carrying the AC response of acceptance or rejection in the whole path is sent back to the ingress, and an accepted flow does not have to wait to start to transmit). Moreover, the scheme assumes that aborting a connection implies that the TCP source will not transmit any packet, and that the accepted TCP connections will share resources fairly as usual. Therefore, the scheme does not have to control whether the packets entering the network belong to an accepted flow or to control the traffic sent by accepted flows to provide isolation, and per-flow (connection) state is not required in any router. Therefore, the scheme relies on TCP sources being well-behaved, as in the "traditional" scheme. Another disadvantage of this scheme is that it is not possible to detect sequences of packets that occur as bursts within persistent TCP connections.

The AC algorithm in [kuma00a] measures the actual occupancy of the link and compares it with a given threshold, and when it is exceeded, new arriving connections are rejected. Specifically, a hysteresis with two thresholds is built to avoid excessive oscillations: connections are rejected when the occupancy exceeds the higher threshold and until the occupancy decreases below a lower threshold. The authors suggest occupancy thresholds of around 90% of the link's capacity. The relationship between the occupancy threshold and the connections' throughput comes from an analytical model, which considers ideal fair resource sharing of a random number of flows, similar to the model used in [ben01b] (see Subsection 2.4.3). For example, the model predicts an average flow's throughput equal to 20% of the link's capacity when the occupancy is around 90%. In the case of [mort00a], the AC algorithm measures the incoming traffic to the link's queue and derives the actual overflow (loss) probability using a statistical model. A new arriving connection is rejected whenever this packet loss probability exceeds a given threshold. In this way a minimum throughput is provided since TCP's throughput is related to packet loss. However, note that both AC algorithms use parameters loosely related to the flow's throughput, and therefore tuning the performance is not easy. Moreover, both AC algorithms do not immediately consider the effect of a recently accepted flow until future measurements take it into account. This takes some

time and therefore a high rate of new arriving flows to a router may cause false acceptances. (However, this has another consequence in a hop-by-hop scheme: the partial acceptance of a flow in a hop, which later is rejected in the following hops, does not prevent other flows from being accepted in this hop; therefore, it does not lead to false rejections).

4.2.3 The scheme for elastic traffic in Cross-Protect

The Cross-Protect architecture [robe04b, kort04a] provides two services, a low jitter and low loss service for real-time flows and a minimum throughput service for elastic flows. The minimum throughput's value is the same for all elastic flows while the peak rate of real-time flows should be smaller than a given value. The guarantees are qualitative. It provides isolation to accepted flows. The user-network interface remains as simple as in the traditional Internet, since implicit ways are used instead of per-flow signaling. The AC is fast. It requires per-flow state and per-flow queuing in all routers. The two basic mechanisms are the following:

- The queue discipline uses the Priority Fair Queuing (PFQ) algorithm [kort04a], which shares the link's capacity fairly between all flows and also gives priority to flows whose peak rate is less than the current link's fair rate. It requires per-flow state in each router.
- The AC scheme is hop-by-hop, based on measurements and does not use signaling. It requires per-flow state in each router. It does not differentiate between elastic and real-time flows or different traffic rates. It ensures that the current priority traffic load is smaller than a given percentage of the link's capacity, and that the fair rate is higher than a given threshold. This threshold is chosen to be higher than the peak rate of expected real-time flows, so that they receive scheduling priority in PFQ queues.

PFQ is an enhancement of the FQ algorithms. Like FQ it shares the link's capacity fairly between all flows, so that each flow receives the max-min fair share and is isolated from other flows. In addition to this, PFQ gives scheduling priority to packets from flows whose peak rate is smaller than the current fair rate, so that these flows experience low jitter and low loss. In this way the requested flow's QoS (real-time or elastic) can be implicitly indicated (without signaling): a flow whose peak rate is smaller than the fair rate is considered a real-time flow; otherwise it is considered to be an elastic flow.

AC and PFQ help each other. AC maintains the fair rate above a threshold, which is chosen to be higher than the expected peak rates of real-time flows. PFQ maintains a list of active flows, which is smaller than the list of accepted flows, and scalability is assured by the fact the number of flows is bounded by AC. PFQ provides two measurements that are used by the AC algorithm: *fair_rate*, an estimation of the rate currently realized by backlogged flows, and *prio_load*, the current load of the traffic receiving scheduling priority.

The AC scheme is hop-by-hop since each router performs a local AC decision (it could also be used as a one-hop scheme on LPs with reservation). It does not use any signaling and therefore it requires per-flow state in each router to detect the new flows. Each router maintains a list of accepted flows in an implicit way. A new flow is indicated to a router by the arrival of its first packet, the router indicates a local acceptance decision of the flow by forwarding this packet or a local rejection decision by discarding it, and the end of the flow is detected when no packet is received within a defined timeout interval. This way has the advantage of not requiring signaling, and in the case of elastic traffic, sequences of packets that occur as bursts within persistent TCP connections can be detected. Per-flow state consists in a flow identifier and the arrival time of the last packet of each flow. A flow is identified by the usual 5-tuple in IPv4 (protocol, source and destination IP addresses and ports) or by the more flexible 3-tuple in IPv6 (flow label, source and destination IP addresses). Specifically, the procedure is the following. For each arriving packet, the list is checked. If the packet belongs to a flow in the

list, it is forwarded and the last packet arrival time of the flow in the list is updated. If the packet does not belong to any flow in the list, an AC decision for the new flow is made. If the flow is accepted, the packet is forwarded and a new entry is added to the list. If the flow is rejected, the packet is discarded. A flow is erased from the list when the time since the last packet arrival exceeds the defined timeout.

The AC scheme does not use any explicit indication of the requested service, neither the QoS (real-time or elastic) nor the traffic parameters (the minimum throughput for elastic or the peak rate for real-time). The AC algorithm does not distinguish between elastic and real-time flows. The traffic parameter of the new arriving flow is implicitly supposed to be a given value, which is defined by the network (the maximum between the following values: the minimum throughput of elastic flows and the possible peak rates of real-time flows). This implicit approach has two important advantages: signaling carrying the flow's traffic parameters is not required, and the blocking probabilities of all flows are equal, independently from their requested traffic rate (see the "trunk reservation" mechanism in [cost242]).

The AC algorithm is based on measurements using the above mentioned *fair_rate* and *prio_load*. The general goal of the AC algorithm is to ensure that the current priority traffic load (*prio_load*) is smaller than a given percentage of the link's capacity, and that the fair rate (*fair_rate*) is higher than the mentioned threshold (i.e, a value higher than the expected peak rates of real-time flows). The detailed algorithm is not yet specified (e.g., the percentage of link's capacity for *prio_load*, or whether the measurements of *fair_rate* and *prio_load* are artificially updated once a flow is accepted in order to establish a reservation immediately), although a recommended threshold for the fair rate is about 1% of the link's capacity.

Note that the AC is fast, as it is made as soon as a new flow arrives, since no signaling packet carrying the AC response (of acceptance or rejection) in the whole path is sent back to the ingress, and an accepted flow does not have to wait to start to transmit. Also note that, as it happens in any hop-by-hop AC scheme, a partial reservation in a hop for a flow (established immediately when it is accepted), which is later rejected in other hops, may prevent other flows from being accepted in this hop (for some time), leading to false rejections. However, in this scheme, since no AC response signaling packet is sent back to the ingress, this situation can last for more time and be worse if a rejected (but partially accepted) flow persists in transmitting (although this is not likely to happen).

4.3 Conclusions

We have studied the main network schemes that have been proposed in the Internet for TCP elastic traffic, with and without AC, focusing on the main characteristics of the service (whether the minimum throughput can be different or is the same for all flows, whether isolation between flows is provided, etc.) and their architecture (the specific traffic conditioning, queue disciplines and AC mechanisms used, the required state, the use of signaling, etc.).

We have studied these network schemes without AC:

- The "traditional" scheme provides the best-effort service, which in combination with TCP rate-adaptive algorithms provides a fair throughput service. Different throughputs and isolation between flows are not provided. It is based only on FIFO and Tail Drop (or RED) queues.
- The schemes based on packet classes can provide different throughputs and isolation between flows. Traffic conditioning mechanisms at the ingress assign each flow's packet to a class (e.g., by comparing the flow's average sending rate and the flow's desired minimum throughput, flow's packets are assigned to an *in* or *out* class) and queue disciplines are

based on classes (e.g., the *out* class has a higher discarding priority than the *in* class). Per-flow state is only kept at the edge while the core remains simple and highly scalable.

– The scheme based on CSFQ provides a fair throughput service (a weighted version is also possible) and isolation between flows. The ingress router estimates the flow's incoming rate and writes it on a label in the packet's header. The CSFQ algorithm in queues discards packets probabilistically, using only the packet's label and aggregated measurements, so that each flow receives the fair rate. Per-flow state is only required at the edge. However, it requires processing and updating the label for each packet in each router, as well as encoding the label in the packet's header.
– The USD scheme provides different throughputs to flows from different users but in an aggregated way. It provides isolation between flows from different users. Each user is assigned a given weight and WFQ in queues share resources between users according to this weight for both the sending and the receiving traffic. It requires per-user state in all routers (it does not require per-flow state).

We have studied the following network schemes with AC:

– The scheme for a throughput service in Corelite provides different minimum throughputs to different flows as well as isolation. The ingress router turns some flows' packets into the so-called (*g* or *p*) markers, so that their rate indicates the minimum throughput, and other flows' packets into *w*-markers, so that their rate indicates the assigned extra throughput. In each router, ordinary and marker packets are scheduled together with FIFO. When congestion is detected *w*-markers are sent back to the ingress router, which then decreases the extra throughput assigned to the flow. The AC scheme is hop-by-hop, per-flow signaling carries the AC request and response, and each router determines the aggregated reservation by measuring the arriving *g* and *p* markers during a given time period. It neither requires per-flow state in the core nor per-flow queuing. However, it requires per-flow signaling, which could result in a high overhead and a rather long duration of the AC phase as well as quite complex management for the markers. Finally there are no details about how TCP flows are defined and identified in the scheme.
– The scheme with an implicit AC for TCP connections provides the same minimum throughput to all flows, which are defined here as TCP connections. It does not provide isolation. The data path is simply based on FIFO queues, and the AC scheme is hop-by-hop (or a one hop scheme on LPs with reservation). It is based on measurements and it does not have signaling. The start of the flow (connection) is indicated to a router through the TCP connection establishment packets: in the case of acceptance, the connection establishment is allowed to proceed by forwarding these packets, and otherwise, it is aborted. Therefore, the AC is fast, as it is made as soon as a new flow arrives. For the local AC decision, the router measures the actual use of resources through a particular parameter, and when it exceeds a given threshold, new connections are rejected. Per-flow (connection) state is not required in the core or at the edge. However, the scheme relies on TCP sources being well behaved. It does not detect sequences of packets that occur as bursts within persistent TCP connections. Tuning the performance is not easy since the parameters measured are loosely related to the flow's throughput. The AC algorithms do not immediately consider the effect of a recently accepted flow until future measurements take it into account. This takes some time, and therefore, a high rate of new arriving flows to a router may cause false acceptances.
– The scheme for elastic traffic in Cross-Protect provides the same minimum throughput to elastic flows, which are defined here as sequences of packets within TCP connections. A service for real-time flows is also provided. The scheme provides isolation. Queues use the PFQ algorithm, which shares the link's capacity fairly between flows, provides isolation and gives priority to flows whose peak rate is less than the current link's fair rate (i.e., for real-time flows). PFQ requires per-flow state. The AC scheme is hop-by-hop (or a one hop scheme

on LPs with reservation), based on measurements and without signaling. The AC requires a per-flow state. A new flow is indicated by the arrival of its first packet. The router indicates a local acceptance decision of the flow by forwarding this packet or a local rejection decision by discarding it. The end of the flow is detected when no packet is received within a defined timeout interval. Therefore, the AC is fast, as it is made as soon as a new flow arrives. The AC algorithm does not differentiate between elastic and real-time flows and the traffic rate of the new arriving flow is supposed to be the maximum possible value. It ensures that the current priority traffic load is smaller than a given percentage of the link's capacity, and that the fair rate is higher than a given threshold (which is chosen to be higher than the peak rate of the expected real-time flows).

From among the different network schemes without AC we have studied, the ones based on packet classes show a good trade-off between the simplicity (per-flow operations are kept at the edge only) and the service characteristics (they allow different throughputs and isolation between flows to be provided). Out of the network schemes with AC we have studied, we found that is of special interest the definition of flow used in the scheme for elastic traffic in Cross-Protect, as it captures the sequences of packets that occur as bursts within persistent TCP connections, as well as the implicit way of detecting the start and end of these flows. Another interesting aspect of some of these schemes with AC is the utilization of implicit ways for indicating the requested service parameters (QoS and traffic), although they achieve this by providing the same minimum throughput to all flows. In all of them the AC is hop-by-hop and based on measurements. However, they require either per-flow signaling in the core, or are not able to provide different throughputs or isolation between flows, or require per-flow state and per-flow queuing in the core.

Chapter 5:

The 1st scheme: a network scheme for TCP elastic traffic with AC using edge-to-edge per-flow measurements in class-based networks

In this chapter we describe our first proposal of a network scheme with AC for TCP elastic traffic. The scheme considers that a flow is a sequence of related packets within a TCP connection, and it is able to provide different minimum throughputs (to flows from different users) and isolation between flows. It considers a multidomain scenario in which each domain has a user-provider agreement (SLA) with each of its neighboring domains. It is built with simple mechanisms without using per-flow state, per flow signaling or per-flow processing in the core. The whole scheme uses a small set of packet classes, specifically, four classes plus the best-effort class, with a different discarding priority assigned to each one. The AC is implicit, edge-to-edge and based on per-flow throughput measurements. The first packets of the flow are used as a probing flow to test if the network throughput in the path is enough to satisfy the flow's request. Then signaling packets carry the per-flow throughput measurement from the egress to the ingress, where the AC decision is made. However, the short-term fluctuations of the sending rate of a "standard" TCP source reduce the performance, and in order to avoid this situation, we propose a modification of TCP's sending algorithms that keeps the short-term sending rate close to the actual average and above a minimum value. Through simulation we show that our 1st scheme with this especially modified TCP guarantees the requested minimum throughput to accepted flows and achieves good utilization of network resources.

The chapter is organized as follows. First, we present the basic features of the scheme and we discuss the reasons behind its architecture. Then we describe the scheme in detail including the modification of the TCP source. After that we present the simulation results we obtained for evaluating the performance. Finally we present the conclusions.

5.1 Introduction

Our goal is to design a network scheme with AC for TCP elastic traffic using simple mechanisms. The general requirements are the following (see Section 1.2):

– It should guarantee the MTS to the maximum possible number of flows, where a flow is defined as a sequence of related packets within a TCP connection. It should be able to provide different minimum throughputs to different users. It should provide isolation between flows. It should be built with simple mechanisms, i.e., with mechanisms (of traffic conditioning, queue disciplines and AC) that reduce the per-flow state, per-flow signaling and per-flow processing as much as possible. It should consider a multidomain scenario in which each domain has a user-provider agreement (SLA) with each of its neighboring domains.

In order to meet these requirements we propose using a network based on packet classes and an implicit and measurement-based edge-to-edge AC. More specifically, the basic features of the architecture of our 1st scheme are the following:

– It uses packet classes, such as the classes in the Diffserv architecture (Subsection 2.3.3) and the set of schemes explained in Subsection 4.1.2. Traffic conditioning at the network ingress assigns each flow's packet to a class and writes a mark in the packet's header that identifies the class (the number of classes is small), according to an agreed traffic profile. Queue disciplines in the network core are based on classes and apply different treatments to packets belonging to different classes. Per-flow state is kept at the edge only while the core remains simple and highly scalable. Moreover, using traffic conditioning and class-based queues allow isolation between flows and different throughputs to different flows to be provided, as well as coexisting with other different network services.
– The AC is edge-to-edge and based on measurements (see Section 3.5). In edge-to-edge AC schemes only edge routers participate in the AC decision, exchange signaling and maintain per-flow state (in order to detect new flows and to isolate the accepted flows by enforcing the agreed traffic profile to their input traffic), while core routers do not maintain either per-flow or aggregate reservation state and do not require signaling. This feature is the main advantage in relation to other AC schemes: in hop-by-hop AC schemes, each router maintains the actual aggregated reservation and performs a local AC decision, and per-flow signaling and/or per-flow state are required; in one-hop AC schemes on LPs with resource reservation, the AC decision is only at the ingress router, but per-path resource reservations require more complex management to avoid that the LPs passing through the same link share resources in a non-appropriate way. Moreover, there are different ways of achieving an edge-to-edge AC scheme, but using measurements plays a central role in all of them. Measurements are advantageous since they can provide a better estimate of the traffic load and increase resource utilization. However, they run the risk that traffic fluctuations may cause eventual QoS degradations and therefore they provide qualitative guarantees.
– The AC is implicit (see Subsection 3.1.3). The start of a flow is detected at the network ingress when its first packet is received. The port numbers and/or a specific mark in the packets are used to indicate that the flow requests the MTS and the desired minimum throughput, in a way specified in the user-provider agreement. Acceptance is indicated simply by providing the MTS, while rejection is indicated by providing the best-effort service, and packets are forwarded and marked accordingly (note that another possible way of indicating rejection is to discard the packet). A list of active flows is maintained at the network ingress, with the flows' identifiers (e.g., IP addresses, protocol and port numbers) and other information, in a similar way to the Cross-Protect architecture (see Subsection 4.2.3). For each arriving packet, its flow's identifier is checked against this list in order to know whether it belongs to a flow in the list or it is the first packet of a new flow. In the second case, an AC decision is made, and then a new entry in the list is added. The flow's entry is removed when the flow is inactive, i.e., when no packet is received within a defined timeout interval. The main advantage of using an implicit AC is that per-flow signaling is not required, and besides, flows that occur as sequences of related packets or bursts within persistent TCP connections can be detected.

– The SLS part of the user-provider agreement defines the following aspects of the service delivery. Firstly, the agreement defines the value of the maximum aggregated throughput the user may ask for, i.e., the user may ask for any throughput for a flow to any destination as long as the aggregated throughput of all its accepted flows does not exceed this contracted maximum value. Secondly, the agreement defines the values of the port numbers and/or the specific mark in the packets that indicate that a flow requests the MTS and the desired minimum throughput, e.g., the same throughput for all, or different according to the application type (ftp, web, etc.) or other.
– The network uses pre-established LPs from ingress points to egress points, such as the LSPs in MPLS architecture (Subsection 2.3.4), when they are used without resource reservation. Once a flow is accepted by our AC, resources in the path are reserved during the flow's lifetime. We use the LPs to pin each flow to a path, so that all flow packets follow the path where the reservation has been made.

Edge-to-edge AC schemes (see Section 3.5) can be based on active measurements, using per-flow or per-aggregate probing, or based on passive measurements. The AC of our 1st scheme has some similarities mainly with the architecture of active measurement-based edge-to-edge AC schemes with per-flow probing, and also with the non-intrusive nature of passive measurement-based edge-to-edge AC schemes.

In active measurement-based edge-to-edge AC schemes with per-flow probing, for each new flow, a special probing flow is generated and sent through the network path. The receiver end-point measures its experienced QoS to estimate if there are enough available resources in the path to satisfy the flow's request. A special packet carries the per-flow measurement to the ingress. An acceptance or rejection decision is made according to the measurement and the resources requested by the new flow. The duration of the AC phase is about one RTT plus the measuring time, after which an accepted flow is allowed to transmit. However, there are several differences between these generic AC schemes and the AC of our 1st scheme. These generic schemes are focused on real-time flows, while our 1st scheme focuses on elastic flows. This requires some adaptations since elastic flows have different traffic characteristics and QoS requirements than real-time flows. Real-time flows have a constant or variable sending rate (which is less variable than elastic flows) and a longer duration, and they require a very small packet loss ratio and small packet delay and jitter. Elastic flows, however, have a very variable sending rate and generally shorter duration, they require a minimum network throughput and they can benefit from extra throughput. In these generic schemes the special probing flow is an extra flow of packets sent before the data packets and the AC decision. However, this is not appropriate for elastic flows due to their generally shorter duration. In the AC of our 1st scheme the first packets of the flow are used as a probing flow to test if the network throughput in a path is enough to satisfy the flow's request. This means that there is no intrusive traffic and that measurements are passive instead of active. This also means that an accepted flow does not have to wait until the end of the AC phase before starting to transmit. Moreover, in these generic schemes the queues use different priorities for scheduling packets (packets of accepted flows have higher priority than packets of flows in the AC phase), while in the AC of our 1st scheme we use different priorities for discarding packets.

Finally, note that the throughput guarantee is qualitative (low loss if the sending rate is not higher than the minimum throughput) because of the measurements, and that the extra throughput comes from the best-effort sharing of the remaining resources between flows.

5.2 Description of the scheme

In this section we describe our 1st scheme in detail. First, we deal with the main parts of the architecture, i.e., the definition of the packet classes, how the AC works and the edge-to-edge

per-flow measurements. After that we explain why the short-term fluctuations of the sending rate of a "standard" TCP source can reduce the performance of the scheme, and in order to avoid this situation, we propose a modification of the sending algorithms of TCP to keep the short-term sending rate close to the actual average and above a minimum value. Then we deal with the interdomain aspects and the operations performed by the edge nodes.

5.2.1 The architecture

When the first packet of a new flow *f* arrives to the network, the flow is assigned to a logical path *LP*, the list of active flows at the ingress is updated, and the requested minimum throughput of the flow $Rreq_{LP}^{f}$ (in [bps]) is obtained from the corresponding user-provider agreement. The AC evaluates whether this minimum throughput requirement can be provided without losing the minimum throughput guaranteed to the accepted flows. If the flow is accepted, it receives the MTS, and if the flow is rejected, it receives the best-effort service.

Our 1st scheme uses five packet classes, each with a different discarding priority. The classes are two A (Acceptance) classes, A_{IN} and A_{OUT}, two R (Requirement) classes, R_{IN} and R_{OUT}, and the BE (Best-Effort) class. The first packets of the flow are used as a probing flow to test if the network throughput in a path is enough to satisfy the flow's request. It works in the following way (Figure 5-1):

- During the AC phase, at the ingress, the flow's packets are marked as R by the traffic meter and the marker: depending on the comparison between the average sending rate and the desired minimum throughput, each flow's packet is classified as in-profile or out-profile, and then *in* packets are marked as R_{IN} and *out* packets as R_{OUT}.
- During the AC phase, at the egress, the throughput of the flow is measured during a time period of duration Tfl_{LP} (in [s]). Then the measured throughput Mfl_{LP}^{f} (in [bps]) is sent to the corresponding ingress through a signaling packet.
- Once the measured throughput Mfl_{LP}^{f} is received at the ingress (the end of the AC phase), it is compared to the requested minimum throughput $Rreq_{LP}^{f}$: the flow is accepted if $Rreq_{LP}^{f} \leq Mfl_{LP}^{f}$, and otherwise it is rejected.
- After the AC phase, at the ingress, if the flow has been accepted, its packets are marked as A (A_{IN} or A_{OUT}, depending on the comparison between the average sending rate and the desired minimum throughput) If it has been rejected, its packets are marked as BE.
- In the output links of all routers there is a FIFO queue (to maintain packet ordering) with discarding priorities for each class in the order (from low to high) A_{IN}, R_{IN}, A_{OUT}, R_{OUT} and BE.

The priorities are chosen so that accepted flows are protected against flows that are in the AC phase. Since the lowest discarding priority class receives the resources first, the next class receives the remaining resources, and so on, this roughly means that the R class packets must be discarded before the A class packets (A < R). Specifically, the scheme requires the flows in the AC phase to be able to get a measurement of the unreserved throughput in the path (A_{IN} < R_{IN} < A_{OUT}). Moreover, the remaining extra throughput is given to accepted flows rather than to flows that are in the AC phase (A_{OUT} < R_{OUT}).

The measurement process of the flow's throughput at the egress is the following. A list of active flows being measured is maintained at the egress. That is, when a new flow arrives, it is added to the list, and when the measurement is finished, it is removed. The measured flow's throughput Mfl_{LP}^{f} is simply the total number of bits of the received flow's packets during the measurement period (since the arrival of the first packet), Bfl_{LP}^{f} (in [bit]), divided by the measurement duration Tfl_{LP}, i.e.,

$$Mfl_{LP}^{f} = \frac{Bfl_{LP}^{f}}{Tfl_{LP}} \quad [\text{bps}]. \tag{5-1}$$

Then a signaling packet carries the throughput measurement to the corresponding ingress. Therefore, the duration of the AC phase is equal to the measurement duration Tfl_{LP} plus the packet RTT (for edge routers). Note, however, that an accepted flow does not have to wait until the end of the AC phase before starting to transmit, since the scheme uses the first packets of the flow as a probing flow.

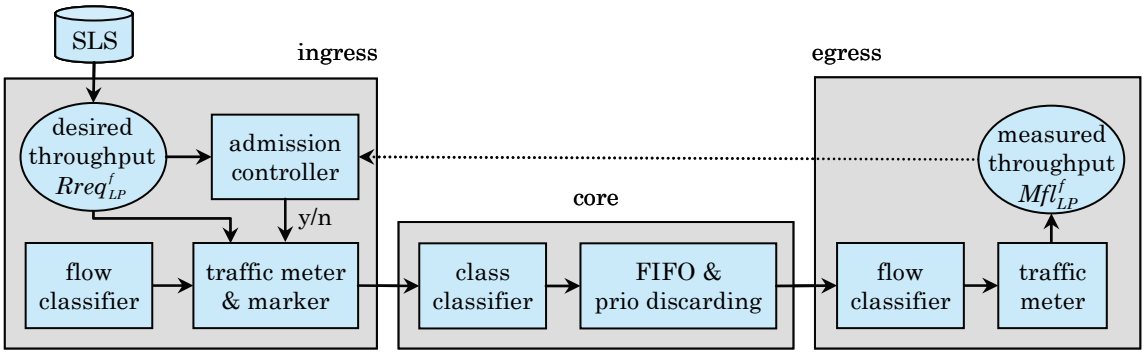


FIGURE 5-1: Functional block diagram of our 1st scheme.

Note that an important point is to decide on the value of Tfl_{LP}. If it is too short, the measurement could be incomplete; but if it is too long, the number of flows that simultaneously compete for certain resources increases and the AC scheme might reject too many flows. In Section 5.3 we study the influence of the measurement duration through simulation. Finally, also note that another possible option for measuring the flow's throughput is to use the received ACK packets at the ingress, without the need for signaling from the egress to the ingress. However, this option cannot always be used since it assumes that the return path reaches the ingress (and also that there are no losses in the return path), a situation that may not occur.

5.2.2 The influence of the short-term fluctuations of the TCP sending rate

The AC of our 1st scheme has some similarities with the active measurement-based edge-to-edge AC schemes with per-flow probing (Subsection 3.5.1). In these schemes reserving resources for an accepted flow is based on occupancy (see Subsection 3.1.3). It is established by a special probing flow (in our scheme we use the first packets of the flow, marked as R_{IN} during the AC phase), which is maintained as long as the flow transmits (following a certain traffic profile), and released when the flow is inactive, without needing signaling packets. This is because measurements reflect all these situations. In this way per-flow state is only required at the edge but not in the core. However, in order for the AC to work properly, sources should never transmit less than the desired minimum throughput. Otherwise, during the AC phase, the measured throughput will be smaller than the requested one and the flow will be rejected; and moreover, after the AC phase, if an accepted flow does not use the guaranteed minimum throughput, other, newly arriving flows in the AC phase will be erroneously accepted, and the throughput allocated to them later on could be incorrect.

When TCP transfers a file and a minimum throughput is available in the network path, the average sending rate during the flow's lifetime is kept above this minimum throughput. However, the traffic sent by a "standard" TCP source is typically very bursty (it sends a number of packets continuously – a burst – according to the actual window and then stops and

waits for the ACKs before going on), i.e., the short-term sending rate fluctuates a lot above and below the average. These fluctuations cause some inaccuracies in the measurements (especially if the measurement duration is short) that can reduce the performance of the scheme. In order to avoid these fluctuations, we propose modifying the sending algorithms of the TCP sources, as we explain in more detail in the next subsection (note that another possibility would be not to modify the sources but rather apply some kind of traffic shaping to the input flow at the ingress, to add dummy packets, etc.). Finally, a consequence of this requirement is that at the ingress the flow's rate should be checked to see if it is always above the minimum throughput, because otherwise the service would be denied.

5.2.3 The modification of the TCP source

We propose modifying the TCP sources in order to avoid the short-term fluctuations of the sending rate and sending at a minimum rate. This modification has some similarities with TCP pacing schemes (see Subsection 2.4.3), in which the source evenly spaces the sending of a window of packets across an entire RTT, instead of sending a burst. Our modified TCP source keeps the short-term sending rate close to the actual average, while this average has a minimum value and is adjusted to network conditions in the same way as a "standard" source. The procedure is the following: instead of a burst, one packet is sent at a certain time Δt (in [s]), which is increased and decreased causing the rate variations; Δt has a maximum value that guarantees the minimum desired rate; the variations of Δt are (inversely) proportional to the variations of the window w (in [bit]) caused by "standard" TCP rate-adaptive algorithms (for the implementation in the simulator, see Section 5.3, we use TCP NewReno). Specifically:

$$\Delta t = K \frac{1}{w} \quad [\text{s}], \qquad w \geq w_{min} \quad [\text{bits}],$$
$$K = w_{min} \frac{pkt\_size}{r_{min}} \quad [\text{s·bit}], \tag{5-2}$$

where *pkt_size* [bit] is the packet length, r_{min} [bps] is the minimum desired rate and w_{min} is the minimum w, with a value to be chosen. Therefore, the short-term sending rate is

$$r = \frac{pkt\_size}{\Delta t} = r_{min} \frac{w}{w_{min}} \quad [\text{bps}], \tag{5-3}$$

i.e., the sending rate is proportional to the window variations of the TCP rate-adaptive algorithms with a desired minimum value r_{min}. Moreover, packet retransmission from the last acknowledged packet is triggered by the usual circumstances (i.e., when the corresponding ACK packet is not received during a timeout period or when three duplicated ACKs of a previous packet are received), and also when the packet sequence number reaches the end value and the corresponding ACK has not been received yet.

5.2.4 Interdomain aspects and edge node operations

In a multidomain scenario the end-to-end service is provided by the concatenation of the service provided by each of the domains of the followed path. Each domain provides the service to the flow in its own way, using its own scheme (e.g., our 1st scheme) and has a user-provider relationship with its neighboring domains according to an agreement (SLA). In the previous subsections we dealt with the intradomain architecture of our 1st scheme. Here we discuss the interdomain aspects and the details of the edge node operations.

The interdomain operation refers to questions such as indicating the required service, identifying the flow that requests the service, indicating service acceptance or rejection, or

other. Figure 5-2 shows a basic scenario with an upstream domain U, our domain O (the one using our 1st scheme) and the downstream domain D. The question here is to determine the interconnection needs of our domain O when acting as a user or a provider. In addition to this, note that our domain O will also have to carry out the interconnection operations required by neighboring domains (e.g., signaling, marking, etc.), which depend on the specific scheme they use. However, we are only discussing the interconnection of our domain with its neighbors here and not the reverse.

For interconnecting our domain O with the upstream domain U (U acts as a user and O as a provider), we use an implicit method without per-flow signaling (see also Section 5.1): the start of a flow is detected at the domain ingress when its first packet is received; the end of a flow is detected when no packet is received within some defined timeout interval; the request of the MTS and the desired minimum throughput is indicated through the port numbers and/or a specific mark in the packets, in a way specified in the user-provider agreement; acceptance is indicated simply by providing the MTS, while the rejection is indicated by providing the best-effort service.

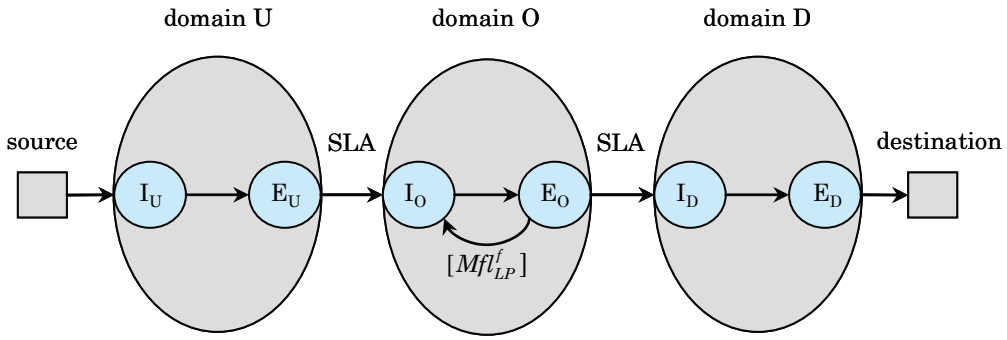


FIGURE 5-2: Interdomain operation of a domain using our 1st scheme with neighboring domains.

For interconnecting our domain O with the downstream domain D (O acts as a user and D as a provider), we use the same implicit way as above. It is worth commenting on the consequences of this approach by studying the following situation: suppose that a given flow is accepted in domain O but is rejected in domain D. Therefore, domain O will keep a useless reservation for this accepted flow, wasting resources that could be used by other flows. However, since the end-to-end decision is rejection, the flow will receive poor service, and the source, after some time of retransmitting and having successive losses, will stop the transmission. Then future measurements will release the reservation. Note that another option that solves this situation more quickly would be to use interdomain signaling packets so that domain D notifies domain O whether it can provide the MTS to the flow or not. With this information, the rejected flow in domain D could also be rejected in domain O, and the useless reservation could be quickly released. However, using per-flow interdomain signaling packets would complicate the scheme.

The operations performed at the ingress of our domain are the following:

- Classifying packets into flows, implicit detection of the start and end of flows and updating the list of active flows.
- Assigning new flows to logical paths.
- Obtaining the desired flow's throughput from the agreement with the upstream domain.
- Checking whether both the new request and the currently accepted ones are within the contracted value specified in the agreement.

– Making the AC decision (from receiving signaling from the egress carrying the per-flow throughput measurement).
– Measuring the corresponding flow's traffic and marking before and after the AC decision.
– Registering services (accepted and rejected flows) provided to the upstream domain.

The following operations are performed at the egress of our domain:

– Classifying packets into flows, implicit detection of the start of flows and updating the list of active flows.
– Measuring the flow's throughput.
– Sending signaling packets to the corresponding ingress for each flow (carrying the per-flow throughput measurement).

5.3 Evaluation of the scheme

In this section we evaluate the performance of our 1st scheme. Firstly, we study its scalability in terms of the number of flows passing through a router. Then we evaluate the performance through simulation. This requires implementing the different parts of the scheme and the modified TCP sources over a simulation platform, choosing the appropriate network topologies and statistical models for generating the traffic, and defining parameters that measure the performance. We describe these different aspects of the simulation in the second subsection. In the third subsection we present the simulation results. We study the influence of some parameters on the performance in several network topologies using different traffic loads consisting of (mainly modified) TCP flows that carry files of varying sizes. We show results for the throughput obtained by the flows as well as for the efficiency of using the links' capacity.

5.3.1 Scalability study

Our 1st scheme involves several operations made by different algorithms in edge and core routers that require using network resources, i.e., memory and processors in routers, and the link's capacity. In this scalability study, we analyze the network resources used by these operations in terms of N, the number of flows passing through a router.

The operations performed by edge routers at the ingress are the following (Subsection 5.2.4): 1) classifying packets into flows, implicit detection of the start and end of flows and updating the list of active flows; 2) assigning new flows to logical paths and obtaining the desired flow's throughput from the agreement; 3) making the AC decision, after receiving signaling packets from the egress carrying the throughput measurement; 4) measuring the input traffic of the flow and the in/out marking before and after the AC decision. The processing time of all these operations is constant with respect to N except in the first one, the lookup in the list of active flows, which can be implemented with algorithms O(log N), considered as scalable in [coul05a]. The use of memory in all these operations is constant with respect to N except in the first one, which is O(N), considered as scalable in [coul05a]. Using link's capacity is not applicable in any of these operations.

The operations performed by edge routers at the egress are the following (Subsection 5.2.4): 1) classifying packets into flows, implicit detection of the start of flows and updating the list of active flows; 2) measuring the flow's throughput; 3) sending the signaling packet (one per flow) to the corresponding ingress with the throughput measurement. The processing time of all these operations is constant with respect to N except in the first one, the lookup in the list of active flows, which can be implemented with algorithms O(log N). The use of memory in all these operations is constant with respect to N except in the first one, which is O(N). Using

link's capacity in all these operations is not applicable except in the third one, which is constant with respect to N.

Finally, the operations performed in core routers are: 1) classifying each packet into a class; 2) packet scheduling and forwarding based on FIFO and packet discarding based on priorities per class. The processing time and using memory in all these operations is constant with respect to N, while using link's capacity is not applicable in the first one and is constant with respect to N in the second one.

5.3.2 Description of the simulation

First we explain the simulation platform we have chosen and describe the different parts of our 1st scheme and the modified TCP that we have implemented over this platform. Then we describe the network topologies and the statistical models for generating the TCP traffic. Finally we define the parameters we use to evaluate the performance of the scheme and we explain how they are obtained from the simulations.

Implementation of the scheme in the simulator

We have chosen the Network Simulator – Version 2 (ns-2) [ns-2] as the simulation platform. It is a discrete event simulator, open-source and object-oriented, with the core written in C++ and an OTcl interpreter as front-end. It is widely used by the networking research community, which continuously enhances the simulator by adding new functionalities. It covers a wide range of networking modules providing support for TCP/IP, including several transport and routing protocols, over wired and wireless networks. The Diffserv module and the set of different TCPs are especially interesting for implementing our scheme.

Firstly, we implemented the functional blocks of our 1st scheme (Subsection 5.2.1) within the Diffserv module of the ns-2 simulator. At the ingress routers there is a sending rate meter, a marker and an admission controller, and at the egress routers there is a throughput meter. A set of these blocks exists for each arriving flow. Moreover, in the output links of all routers there is a single FIFO queue with priority discarding according to the packet classes. Secondly, we implemented the modified TCP source explained in Subsection 5.2.3 (the other "standard" TCP sources we use were already available in the simulator), and also the user's impatience in TCP sources. Specifically, we implemented the following:

- The FIFO queue with priority discarding for the five packet classes (in the order, from low to high, A_{IN}, R_{IN}, A_{OUT}, R_{OUT} and BE), which drops a packet with the highest discarding priority when there is no available buffer space.
- The throughput meter starts to measure when the first packet of the flow arrives, then it simply counts the total received bytes during the chosen measurement duration. Finally, it notifies the ingress router of the throughput measurement Mfl_{LP}^{f} (with the corresponding packet delay).
- The sending rate meter and the marker for the modified TCP source are based on the token bucket algorithm, which is characterized by two parameters, the rate in [bps] and the burst size in [bytes] (we use a burst size of 2 packets and a rate equal to the desired minimum throughput). For the different "standard" TCP sources, the sending rate meter is based on the TSW algorithm and the marker is probabilistic (see Subsection 4.1.2). The TSW is characterized by the *win_lenght* parameter (we use the flow's RTT) and the marker is characterized by the average rate's threshold (we use the desired minimum throughput). In both cases, the marker uses five marks to identify the five packet classes A_{IN}, R_{IN}, A_{OUT}, R_{OUT} and BE: during the AC phase, packets are marked as R_{IN} or R_{OUT}, and after the AC decision, as A_{IN} or A_{OUT} if the flow is accepted, or as BE if it is rejected.

– The admission controller compares the desired minimum throughput $Rreq_{LP}^{f}$ with the received measured throughput Mfl_{LP}^{f} and performs the AC decision.
– The modified TCP source keeps the short-term sending rate close to the actual average and above a minimum value. It is based on TCP NewReno. The chosen value of the w_{min} parameter is 4 packets and the r_{min} parameter is equal to the desired minimum throughput.
– All TCP sources consider the users' impatience, that is, the source stops sending if the transfer time is too high.

Appendix A provides more details about the implementation in the ns-2 simulator.

Network topologies and statistical traffic model

We use several network topologies (Figure 5-3) with different numbers of hops and logical paths. Topology 1 has a bottleneck link of 2 Mbps and two logical paths between pairs of ingress-egress edge routers (e0e3, e2e3), both with the same packet RTT (around 165 ms for hosts and 84 ms for edge routers, without the queuing time). The length of the output queue is 50 packets (around one RTT link's capacity product). Topology 2 is equal to Topology 1 except that the packet RTTs of the two logical paths are different. In Topology 3 there are three logical paths, e0e8 with two hops, and e1e5 and e3e7 with one hop, and two bottleneck links of 2 Mbps. Topology 4 has four logical paths, e0e6, e2e6, e3e6 and e5e6, with similar packet RTTs but a different number of hops, and two bottleneck links of 2 and 4 Mbps. Finally, Topology 5 has six logical paths, e0e9, e2e9, e3e9, e5e9, e6e9 and e8e9, with similar packet RTTs but a different number of hops, and three bottleneck links of 2, 4 and 6 Mbps.

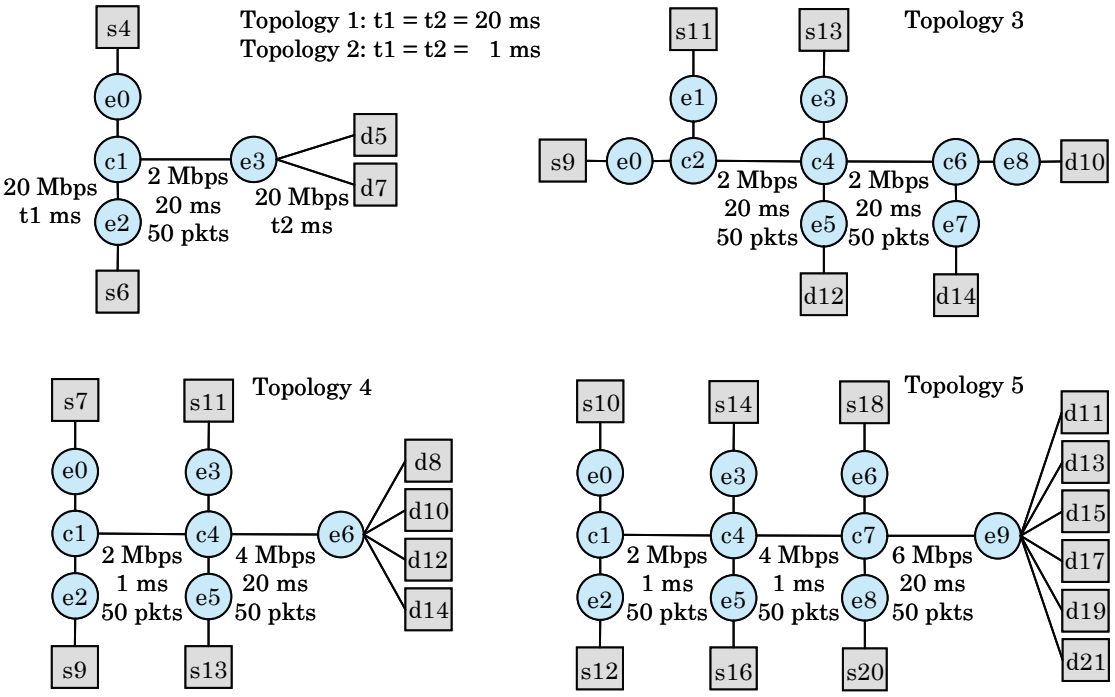


FIGURE 5-3: Network Topologies 1, 2, 3, 4 and 5, indicating the link's capacity and propagation delay, and the length of queues.

We use our modified TCP source, different "standard" TCP sources (TCP NewReno and TCP SACK) and the TCP Westwood source (see Subsection 2.4.3). We generate TCP flows that carry a single file from an ingress point to an egress point through a logical path. Each flow is

characterized by the file size and the starting time. File sizes are obtained from a Pareto distribution that approximates reasonably well the heavy tail behavior of the file size distribution observed in measurements (see Subsection 2.4.4). In all simulations the Pareto constant (or tail parameter) is 1.1 and the minimum file size is 10 packets (the packet length is 1,000 bytes). With these parameters the distribution has an infinite variance. In a typical simulation, after generating the values (about 10,000 flows in each simulation), we observed that more than 50% are below 20 packets, the average σ (in [bit]) is about 74 packets and the maximum reaches 19637 packets. Starting times are obtained from a Poisson arrival process, which is a simple and useful model that has proved to be valid for more realistic scenarios (see Subsection 2.4.4). This process is characterized by the parameter λ (in [flow/s]), i.e., the average number of arrivals per second, which we vary to study different traffic loads. Using all these statistical distributions, the average offered traffic load from an ingress point to an egress point through a logical path is equal to $\lambda\sigma$ bps.

For each logical path we generate the same quantity of offered traffic load, which we vary from 0.05 Mbps to 3 Mbps in steps of 0.3 Mbps, in order to study underloading and overloading situations in the links. In the simulations, we usually choose 90 Kbps as the value of the desired minimum throughput for all TCP flows (which means, for a document of 10 packets, around 10 Kbytes, a minimum transfer time of around 0.9 s) and the user's impatience is twice the desired file transfer time (which means, for 90 Kbps of desired throughput, a throughput of approximately 45 Kbps is obtained).

Performance parameters

We have analyzed the results of the simulation to obtain information about each TCP flow, such as the following: if the flow completes the file transfer or is aborted by the impatient user, the obtained throughput, the AC decision, the number of sent packets, the number of duplicated packets and others. The throughput of the flow is calculated by averaging it over the flow's lifetime, i.e., the total received packets divided by its lifetime.

The goal of the AC is to provide the desired minimum throughput to the maximum possible number of flows. Therefore, to evaluate the performance of our scheme we study the utilization of network resources as well as the throughput obtained by flows. We consider the satisfied flows, which are defined as the ones that complete the file transfer and get at least 95% of the desired minimum throughput (e.g., 85.5 Kbps for 90 Kbps). We obtain the three following performance parameters for each logical path:

- The average "total" satisfied traffic load, which is the aggregated throughput of all satisfied flows, taking into account the minimum and the extra throughput.
- The average "minimum" satisfied traffic load, which takes into account only the minimum throughput.
- The average throughput of satisfied flows (note that it will always be a value above 95% of the desired minimum throughput).

The first parameter measures the use of resources by the in-profile and the out-profile traffic of satisfied flows, the second one measures the use of resources by only the in-profile traffic of satisfied flows, and the third one measures how much extra throughput the satisfied flows get.

The average throughput of satisfied flows is simply obtained by averaging the set of throughput values of satisfied flows. The average total and minimum satisfied traffic loads require more elaborated processing: from the simulator traces, we filter the packets of the aggregation of satisfied flows, distinguishing between the *in* and *out* packets according to their mark, and from that, we calculate the two corresponding time averages. All these average values are obtained by averaging over the simulation time, but without considering the initial

period of the simulation transient phase. The simulation length is chosen so that after this initial transient phase a minimum of 10,000 flows is generated. We make 10 independent replications of each simulation, where independence is accomplished by using different seeds of the random generator (we use the combined multiple recursive generator proposed by L'Ecuyer, with C code extracted from [law00a]). From the 10 independent results we estimate the mean value by computing the sample mean and the 95% confidence interval [law00a].

5.3.3 Simulation results

In this subsection we present the simulation results we have obtained for evaluating the performance of our 1st scheme. We study the influence of some parameters and we show results for the throughput obtained by the flows as well as for the efficiency in using the links' capacity. Specifically, the different "cases" we present are the following:

- 1st case (Figures 5-4 and 5-5). We compare our 1st scheme using the modified TCP, our 1st scheme using other TCPs (the "standard" versions TCP NewReno and TCP SACK, and also TCP Westwood), and the "traditional" scheme (best-effort service) using TCP NewReno.
- 2nd case (Figure 5-6). We analyze in detail the throughput provided by our 1st scheme with the modified TCP to verify that it guarantees the requested minimum throughput to the accepted flows.
- 3rd case (Figure 5-7). We analyze in detail the throughput provided by the "traditional" scheme (best-effort service).
- 4th case (Figure 5-8). We study the influence of the measurement duration Tfl_{LP} on the performance of our 1st scheme.
- 5th case (Figure 5-9). We study the ability of our 1st scheme to provide different minimum throughputs to flows from different users.
- 6th case (Figure 5-10). We study the ability of our 1st scheme to provide isolation between flows.
- 7th case (Figure 5-11). We study the influence of the packet RTT on the performance of our 1st scheme.
- 8th case (Figure 5-12). We study the fairness in the sharing of resources between logical paths achieved by our 1st scheme.
- 9th case (Figures 5-13, 5-14 and 5-15). We study the influence of the number of hops of the logical paths on the performance of our 1st scheme.

In the 1st case we compare our 1st scheme using the modified TCP, our 1st scheme using other TCPs (the "standard" versions TCP NewReno and TCP SACK, and also TCP Westwood), and the "traditional" scheme (best-effort service) using TCP NewReno. The results are shown in Figures 5-4 and 5-5. The desired minimum throughput for all TCP flows is 90 Kbps. We use Topology 1, with measurement durations (in our 1st scheme) for each logical path $Tfl_{03} = Tfl_{23} = 0.2$ s. In the figures we show the results for the satisfied traffic of the "bottleneck" link, i.e., the sum of the two values of the total satisfied traffic of each logical path, the sum of the two values of the minimum satisfied traffic of each logical path and the average between the two values of the average throughput of satisfied flows of each logical path, all three versus the offered traffic load of each logical path, from 0 to 3 Mbps (therefore, the traffic load offered to the link is the sum of the offered traffic load of each logical path, i.e., from 0 to 6 Mbps). Note that, ideally, if the maximum utilization were achieved, the total and the minimum satisfied traffic load would be similar and around 2 Mbps and the average throughput of satisfied flows would be around the desired minimum throughput of 90 Kbps.

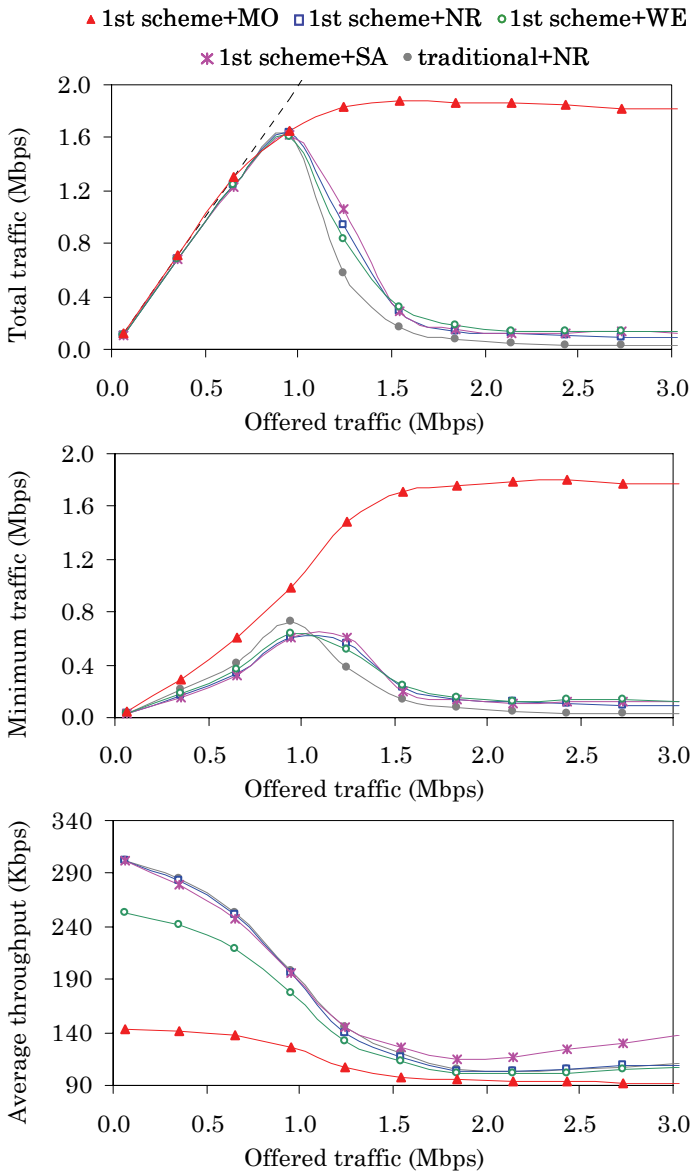


FIGURE 5-4: Comparison of our 1st scheme with the modified TCP (MO), with TCP NewReno (NR), with TCP Westwood (WE) and with TCP SACK (SA), and the "traditional" scheme (best-effort service) with TCP NewReno (NR), in Topology 1 (Tfl_{03} = Tfl_{23} = 0.2 s).

The results in Figure 5-4 show that our 1st scheme achieves a much better performance than the "traditional" scheme when the modified TCP is used and a similar performance when the other TCPs are used. In underloading, when resources are enough to satisfy all flows, all schemes achieve the same value for the total satisfied traffic load, which at the same time is equal to the offered traffic load (the dashed line indicates the equality). In overloading, our 1st scheme with the modified TCP achieves high values of the total and the minimum satisfied traffic load, which stay almost constant (around 1.85 Mbps and 1.75 Mbps respectively) for the entire range of offered traffic load. Our 1st scheme with any of the other TCPs achieves utilization tending to zero due to the effect of the short-term fluctuations of the sending rate on the measurements (see Subsection 5.2.2). The "traditional" scheme (with TCP NewReno) also achieves utilization tending to zero in overloading due to the absence of AC. Moreover, as expected, in all schemes the average throughput of satisfied flows decreases for high values of

the offered traffic load because the unreserved resources decrease (the final value of our 1st scheme with the modified TCP source is around 93 Kbps). Finally, note that in Figure 5-4 we show the sample mean values of the three parameters (for all the schemes), while in Figure 5-5 we show the sample mean values together with the 95% confidence intervals (for our 1st scheme with the modified TCP source), calculated as we explained in the previous subsection.

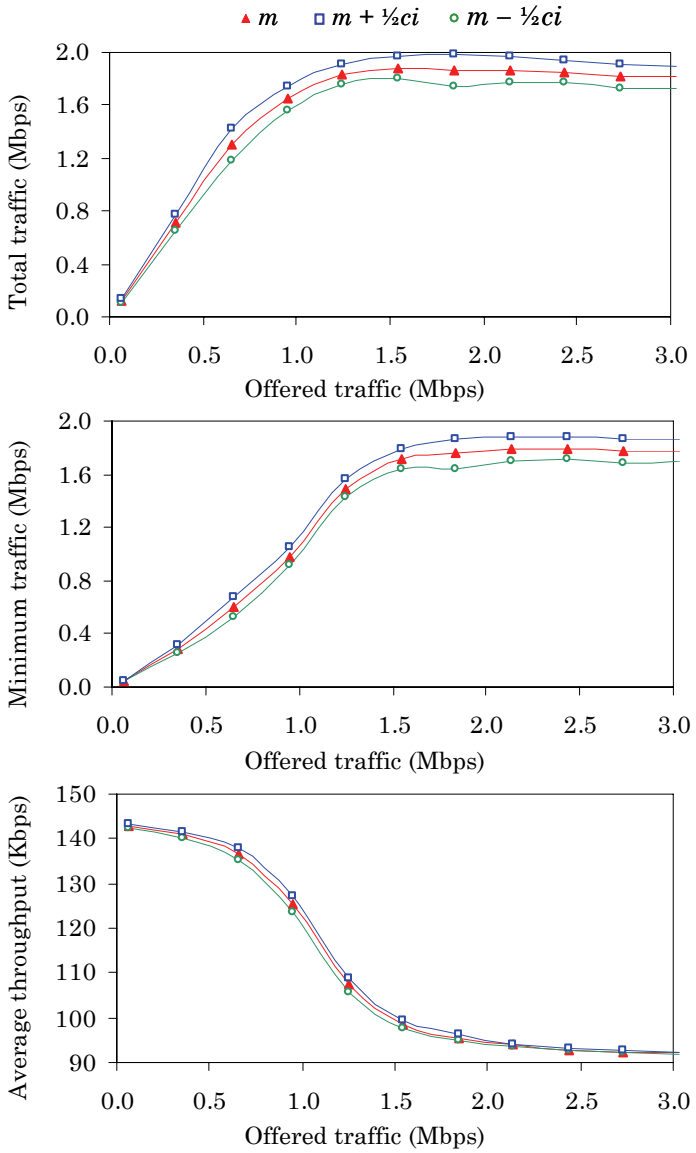


FIGURE 5-5: Sample mean (m) and 95% confidence intervals (ci) of the performance parameters obtained with our 1st scheme and the modified TCP in Topology 1 (Tfl_{03} = Tfl_{23} = 0.2 s).

In the 2nd case, we analyze in detail the throughput provided by our 1st scheme with the modified TCP to verify that it guarantees the requested minimum throughput to the accepted flows. Some of the results obtained are shown in Figure 5-6. We use Topology 1, with measurement durations Tfl_{03} = Tfl_{23} = 0.2 s and a desired minimum throughput of 90 Kbps for all TCP flows. From the simulation results, we have obtained the percentage of accepted flows that complete the file transfer, the percentage of accepted flows that are satisfied (the ones that complete the file transfer and obtain at least the threshold throughput: 95% of the desired

minimum throughput, i.e., 85.5 Kbps), and the frequency distribution of the throughput of the accepted flows to determine how far the accepted and non-satisfied flows are from the threshold throughput. From this analysis we conclude the following:

- All accepted flows complete the file transfer.
- A high percentage of accepted flows are satisfied.
- A high percentage of the accepted and non-satisfied flows achieve a throughput close to the threshold throughput.

The graph at the top of Figure 5-6 shows percentages over the accepted flows versus the offered traffic load of each logical path. 100% of the accepted flows are satisfied for a large range of offered traffic loads and this percentage decreases slowly for high values. The graph at the bottom shows the frequency distribution of throughput of accepted flows (the dashed line indicates the threshold throughput) when the offered traffic load of each logical path is 3 Mbps (in total, 6 Mbps), i.e., in the worst case. Although about 29% of flows are not satisfied, a throughput smaller than 76.5 Kbps is only achieved by 2% of flows, which is a very small percentage.

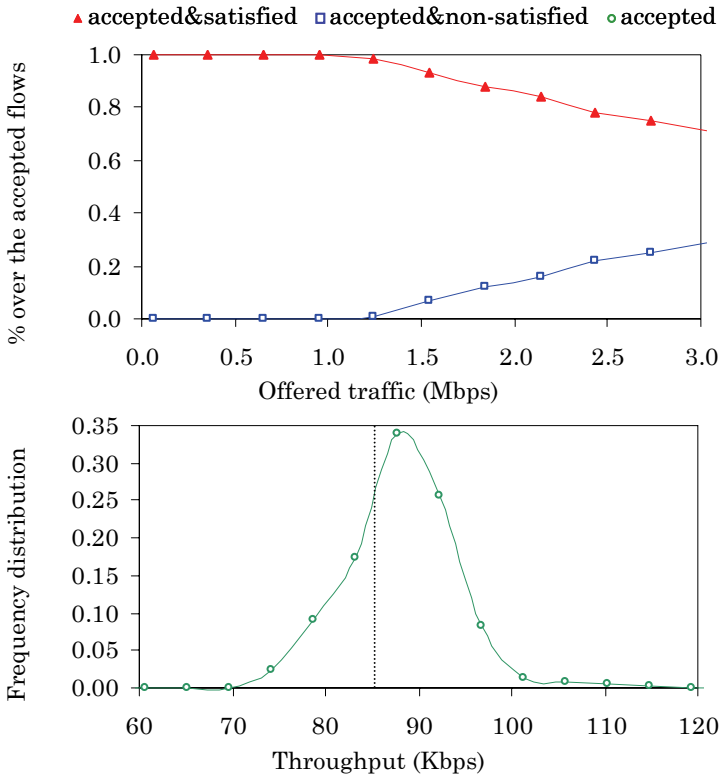


FIGURE 5-6: For our 1st scheme and the modified TCP in Topology 1 (Tfl_{03} = Tfl_{23} = 0.2 s), at the top, percentage of satisfied flows and non-satisfied flows over the accepted flows versus the offered traffic load, and at the bottom, frequency distribution of throughput of accepted flows when the offered traffic load of each logical path is 3 Mbps (in total, 6 Mbps).

In the 3rd case, we analyze in detail the throughput provided by the "traditional" scheme (best-effort service). The results are shown in Figure 5-7. We use Topology 1, TCP NewReno and a 90 Kbps desired minimum throughput for all TCP flows. For each flow we studied whether it completes the file transfer ("completed") and whether it gets at least the threshold throughput ("achieved"). In the figure we show the corresponding percentages over all flows versus the offered traffic load of each logical path. As expected, in underloading, all flows are

satisfied ("completed" and "achieved"), while in overloading the majority of flows do not complete the file transfer ("non-completed" and "non-achieved"), and the few flows that do complete the file transfer get a throughput smaller than the threshold throughput ("completed" and "non-achieved").

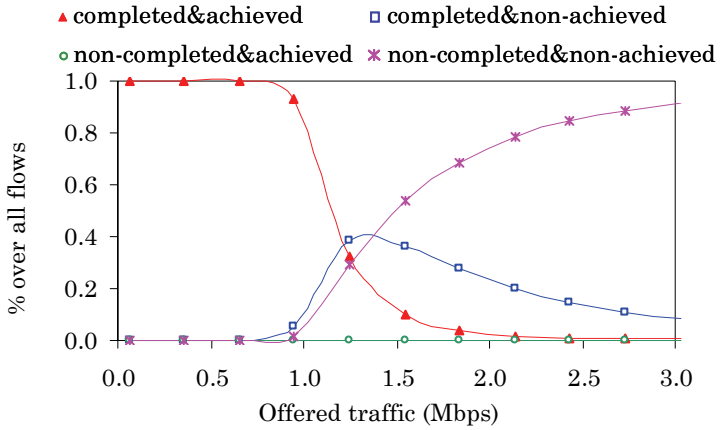


FIGURE 5-7: For the "traditional" scheme (best-effort service) and TCP NewReno in Topology 1, percentages over all flows of "completed" and "achieved" flows, versus the offered traffic load.

In the 4th case, we study the influence of the measurement duration Tfl_{LP} on the performance of our 1st scheme. The results are shown in Figure 5-8. We use the modified TCP, a 90 Kbps desired minimum throughput for all TCP flows, Topology 1 and equal measurement duration for both logical paths $Tfl_{03} = Tfl_{23}$. In the figure we show the three performance parameters for the satisfied traffic of the "bottleneck" link versus the offered traffic load of each logical path. We use five different measurement durations, 0.04 s, 0.2 s, 0.5 s, 0.9 s and 1.3 s (note that the packet inter-arrival time for a CBR flow with packet length of 1,000 bytes and rate of 90 Kbps would be about 0.09 s). The measurement duration of 0.2 s achieves the best performance, when Tfl_{LP} is shorter (0.04 s) the performance is very bad, and when it is longer (0.5 s, 0.9 s and 1.3 s) the performance in overloading is worse and it gets worse as Tfl_{LP} increases.

The reason for the behavior of the curves in Figure 5-8 is the following. When Tfl_{LP} is too short, the measurements are incomplete and wrong, and therefore the majority of flows are erroneously rejected (they receive the best-effort service) and the performance is similar to the one obtained with the "traditional" scheme. When Tfl_{LP} is too long, our AC scheme rejects too many flows in situations where a lot of flows simultaneously meet during the AC phase (the "thrashing" problem experienced by other AC schemes; see, e.g., Subsections 3.1.2, 3.3 and 3.5.1). Imagine a situation in which the AC phases of a set of flows overlap in time, and that these flows have a common bottleneck link, i.e., the total desired minimum throughput of the flows is greater than the available throughput in the link (the total R_{IN} traffic exceeds the available throughput). This available throughput is then shared between the flows proportionally to their desired minimum throughputs, and the measured throughput of each flow is smaller than the desired one. The AC rejects all these flows, and some decisions are therefore erroneous. Specifically, the number of affected flows depends on how strong the overlapping of the AC phases is, since the more the flows overlap, the more flows are erroneously rejected. An increase in the value of Tfl_{LP} causes an increase in the number of flows that overlap and also in the number of erroneous AC decisions, and therefore the performance gets worse. Moreover, an increase in the flow's arrival rate λ also causes an increase in the number of flows that overlap, so the same reasoning explains the decreasing tendency of the total and minimum satisfied traffic load when the offered traffic load

increases. In conclusion, our 1st scheme performs better using a measurement duration Tfl_{LP} that is short; however, there is a limit, because if it is too short, the measurements are wrong.

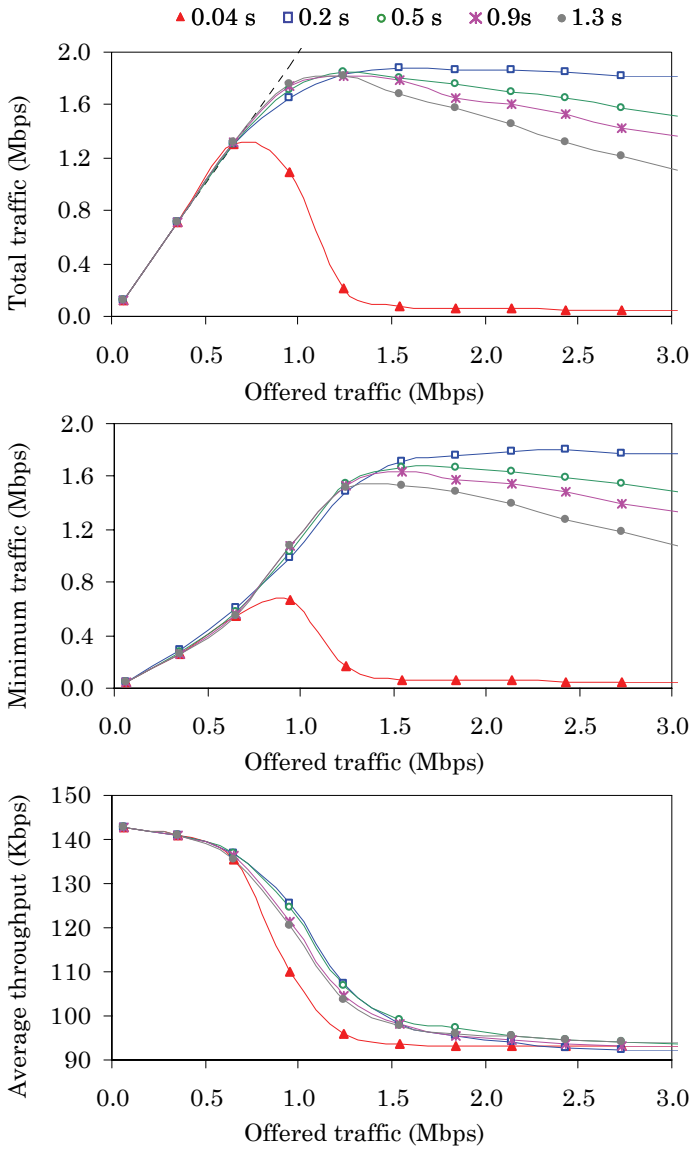


FIGURE 5-8: The influence of the measurement duration Tfl_{LP} on the performance of our 1st scheme and the modified TCP, in Topology 1 (Tfl_{03} = Tfl_{23}).

In the 5th case, we study the ability of our 1st scheme to provide different minimum throughputs to flows from different users. The results are shown in Figure 5-9. We use the modified TCP and Topology 1, with measurement durations Tfl_{03} = Tfl_{23} = 0.2 s. We consider two users, A and B, with desired minimum throughput of 90 and 135 Kbps respectively. User A sends flows through the logical path e0e3 and user B through e2e3. For each user the offered traffic load is the same and varies from 0 to 3 Mbps. In the figure we show the three performance parameters for each user (and also for all together) versus the offered traffic load of each user. As expected, our 1st scheme achieves the desired different average throughputs (the final values are about 94 and 139 Kbps respectively), which is a consequence of the differentiation between the *in* and *out* traffic of the flow. Moreover, there are some small

differences in the sharing of the link between the two users. These differences would mean that the AC accepts more flows demanding a small minimum throughput than flows demanding a large one. However, this is a similar behavior to that observed in other AC schemes where flows demanding more resources experience a higher blocking probability (see, e.g., Subsections 3.3 and 3.5.1).

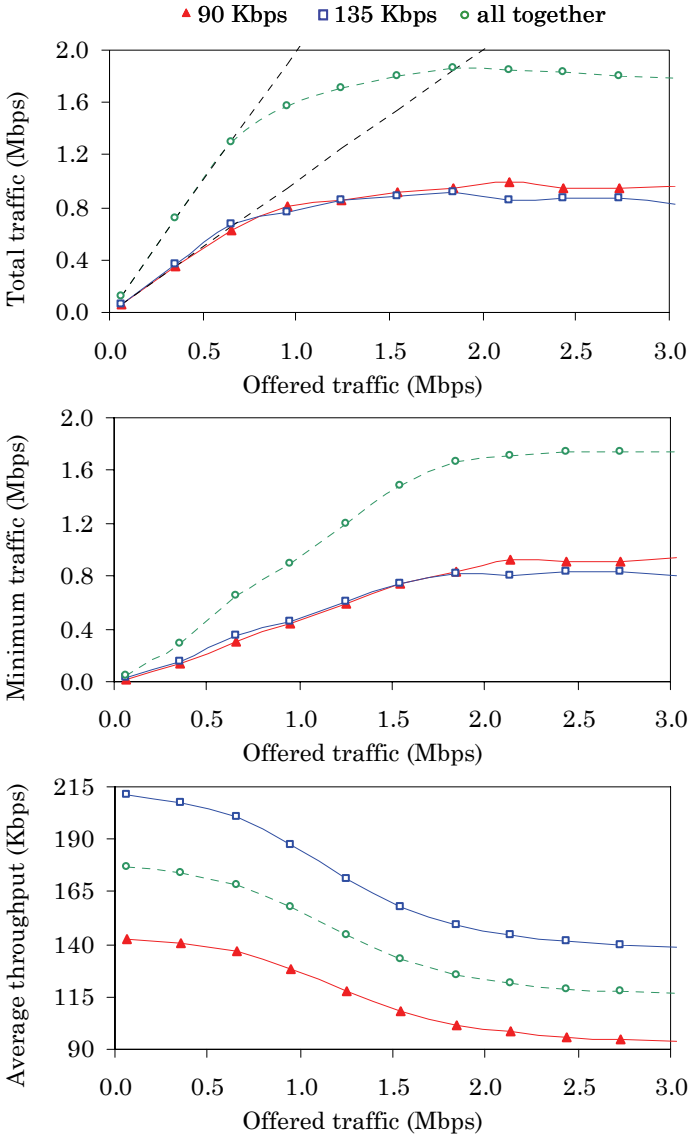


FIGURE 5-9: For our 1st scheme and the modified TCP, results for two users asking for different minimum throughputs, 90 Kbps and 135 Kbps, in Topology 1 (Tfl_{03} = Tfl_{23} = 0.2 s).

In the 6th case, we study the ability of our 1st scheme to provide isolation between flows, so that flows sending more traffic than their allocated throughput do not damage well-behaved flows that do send according to their allocated throughput. The results are shown in Figure 5-10. We use the modified TCP and Topology 1, with measurement durations Tfl_{03} = Tfl_{23} = 0.2 s. We send TCP flows together with CBR flows. The desired minimum throughput of all flows (TCP and CBR) is 90 Kbps, but CBR flows have a constant rate of 135 Kbps while TCP flows adjust the sending rate as usual. CBR flows are generated in a similar way as TCP flows: the

starting times are obtained from a Poisson arrival process (characterized by λ flow/s, the average number of arrivals per second), and the time duration is *Fs*/*R*, where *R* is the flow's sending rate (135 Kbps), and *Fs* is a "file size" (in [bit]) obtained from a Pareto distribution with the same parameters used for the TCP flows. As a consequence, the offered traffic load of the set of CBR flows is again λσ bps. We send TCP flows through the logical path e0e3 and CBR flows through e2e3. For each type of flows the offered traffic load is the same and varies from 0 to 3 Mbps. In the figure we show the three performance parameters for each type of flow (and also for all together) versus the offered traffic load of each type of flow.

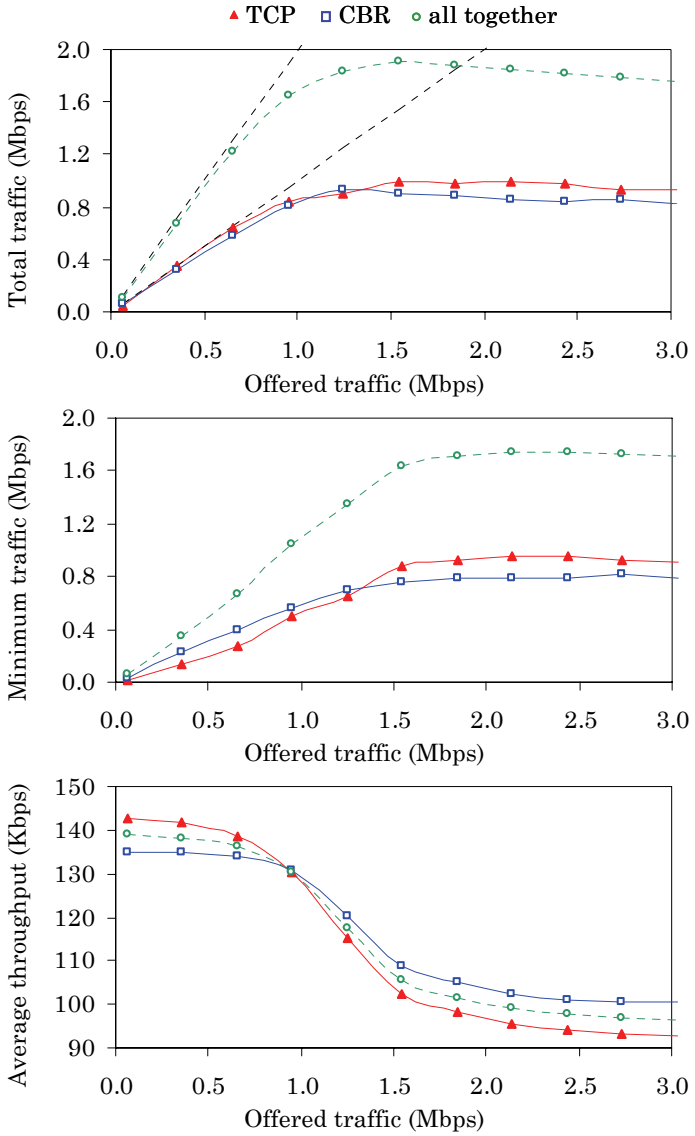


FIGURE 5-10: For our 1st scheme and the modified TCP, isolation of TCP flows against CBR flows of 135 Kbps, when the desired minimum throughput of TCP and CBR flows is 90 Kbps, in Topology 1 (Tfl_{03} = Tfl_{23} = 0.2 s).

The results in Figure 5-10 shows that the accepted TCP and CBR flows achieve the desired minimum throughput even though CBR flows are constantly sending at a higher rate. Again this is because our 1st scheme differentiates between the *in* and *out* traffic of the flow. Moreover, there are some differences in the sharing of the link between the two types of flows

as well as in the extra throughput. This is because the unreserved resources are shared fairly between only CBR flows because they do not adjust the sending rate like TCP, but with a maximum per-flow throughput of 135 Kbps, and the rest is shared fairly between TCP flows.

In the 7th case, we study the influence of the packet RTT on the performance of our 1st scheme. The results are shown in Figure 5-11. We use the modified TCP, a desired minimum throughput for all TCP flows of 90 Kbps, and Topology 2, where the packet RTT for the logical path e0e3 is larger than the one for the logical path e2e3 (specifically, the RTT without taking into account the queuing time is, for e0e3, around 165 ms for hosts and 84 ms for edge routers, and for e2e3, 89 ms and 49 ms respectively; the maximum queuing time is 200 ms). The measurement durations are $Tfl_{03} = Tfl_{23} = 0.2$ s. In the figure we show the three performance parameters for each logical path (and also for all together) versus the offered traffic load of each logical path.

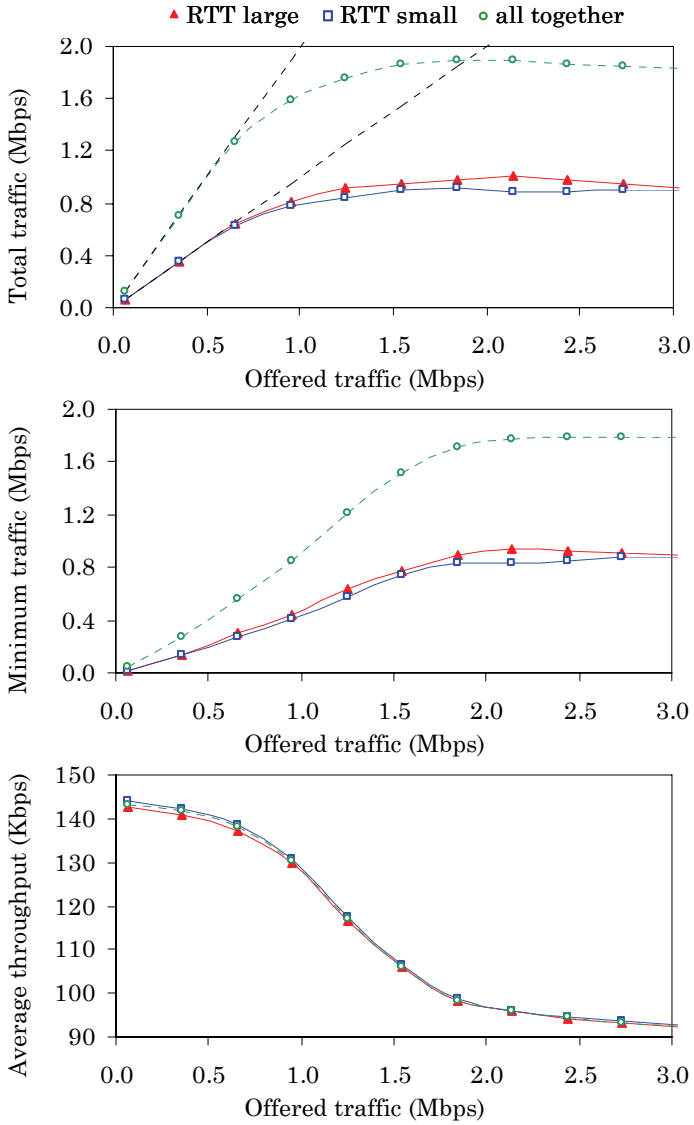


FIGURE 5-11: The influence of the packet RTT on the performance of our 1st scheme with the modified TCP, in Topology 2 ($Tfl_{03} = Tfl_{23} = 0.2$ s).

The results in Figure 5-11 show that both logical paths achieve similar utilization and therefore the performance does not depend on RTT. Also note that the average throughput of satisfied flows is similar for both logical paths, which seems to indicate that the modified TCP does not exhibit the traditional RTT unfairness of the "standard" TCP (see Subsection 2.4.3).

In the 8th case, we study the fairness in the sharing of resources between logical paths. The results are shown in Figure 5-12. We use the modified TCP, Topology 1, measurement durations $Tfl_{03} = Tfl_{23} = 0.2$ s and a desired minimum throughput for all TCP flows of 90 Kbps. We keep the offered traffic load of the logical path e0e3 constant at 1 Mbps and we vary the offered traffic load of the logical path e2e3 from 0 to 3 Mbps. In the figure we show the three performance parameters for each logical path versus the offered traffic load of the logical path e2e3. We expect that the 2 Mbps of the bottleneck link will be shared between the two logical paths proportionally to their respective offered traffic load (this expected result is indicated by the dashed lines). Note that the simulation results are quite close to the expected results.

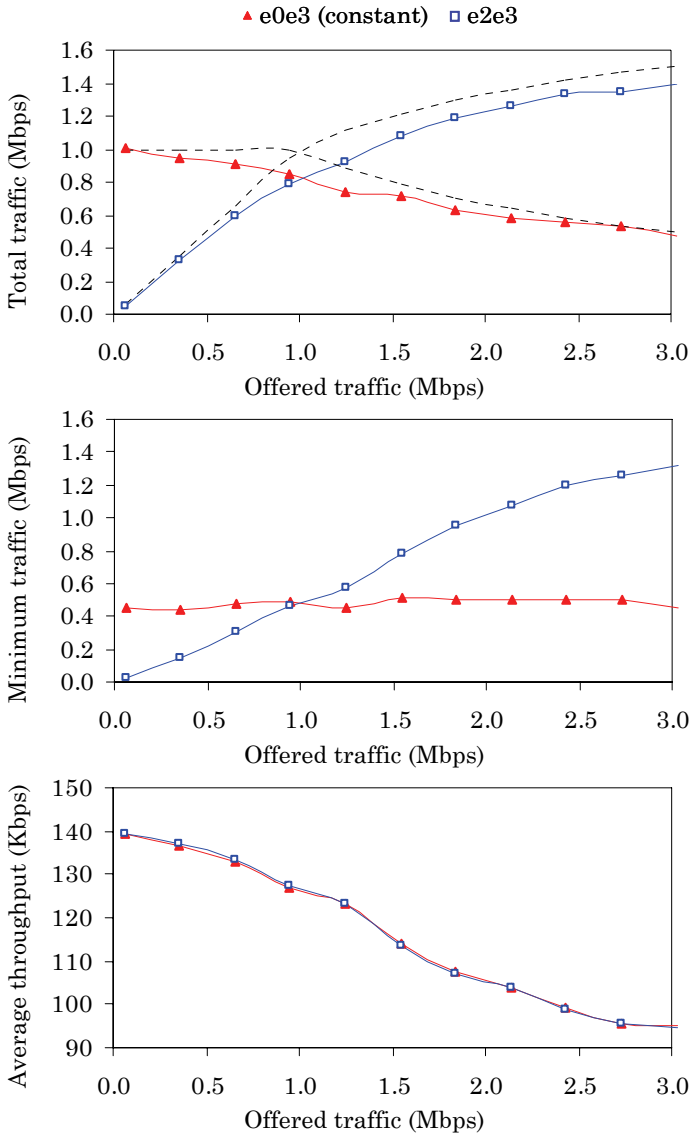


FIGURE 5-12: For our 1st scheme and the modified TCP, sharing between logical paths in Topology 1 ($Tfl_{03} = Tfl_{23} = 0.2$ s). The offered traffic load of e0e3 is constant and equal to 1 Mbps.

In the 9th case, we study the influence of the number of hops of the logical paths on the performance of our 1st scheme. We use three different topologies, 3, 4 and 5, and we show the results in Figures 5-13, 5-14 and 5-15 respectively. We use the modified TCP, a desired minimum throughput of 90 Kbps and the following measurement durations: in Topology 3, $Tfl_{15} = Tfl_{37} = 0.2$ s (1 hop) and $Tfl_{08} = 0.4$ s (2 hops); in Topology 4, $Tfl_{36} = Tfl_{56} = 0.2$ s (1 hop) and $Tfl_{06} = Tfl_{26} = 0.4$ s (2 hops); in Topology 5, $Tfl_{69} = Tfl_{89} = 0.2$ s (1 hop), $Tfl_{39} = Tfl_{59} = 0.4$ s (2 hops) and $Tfl_{09} = Tfl_{29} = 0.5$ s (3 hops). In the figures, for clarity, we do not show the results for each logical path (since the number of logical paths is large), but we combine the results of the logical paths with the same number of hops into a single one. We show, for the three performance parameters, the average values between the pair of logical paths with the same number of hops (1, 2 or 3 hops), all three versus the offered traffic load of each logical path (except in Figure 5-13 for the logical path e0e8, because it is the only one with 2 hops in Topology 3).

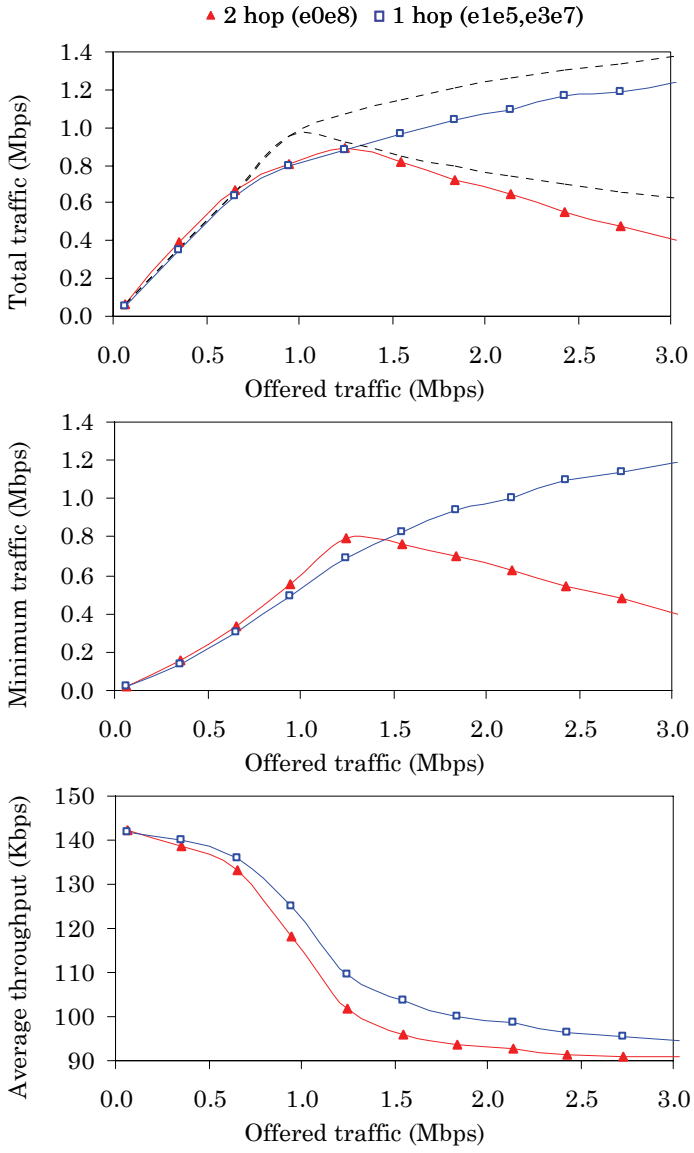


FIGURE 5-13: The influence of the number of hops on the performance of our 1st scheme with the modified TCP, in Topology 3 ($Tfl_{08} = 0.4$ s, $Tfl_{15} = Tfl_{37} = 0.2$ s).

The results show that in congestion, the logical paths with fewer hops obtain a greater utilization than the logical paths with more hops. Logical paths with multiple hops obtain a smaller value of satisfied traffic load since flows passing through multiple congested links experience a higher blocking probability. However, this behavior is similar to the behavior observed in other AC schemes (see, e.g., Subsections 3.3 and 3.5.1). In order to check this similarity, we have compared the results obtained by our 1st scheme with the ideal results that would be obtained by a classical hop-by-hop AC scheme. We obtain this theoretical result taking into account that in our topologies the blocking probability in each core link is the same and that the offered traffic load for each logical path is also the same. These ideal results are represented by dashed lines in the curves of the total satisfied traffic load. Note that the simulation results for the three topologies follow the ideal results, although the total utilization is lower.

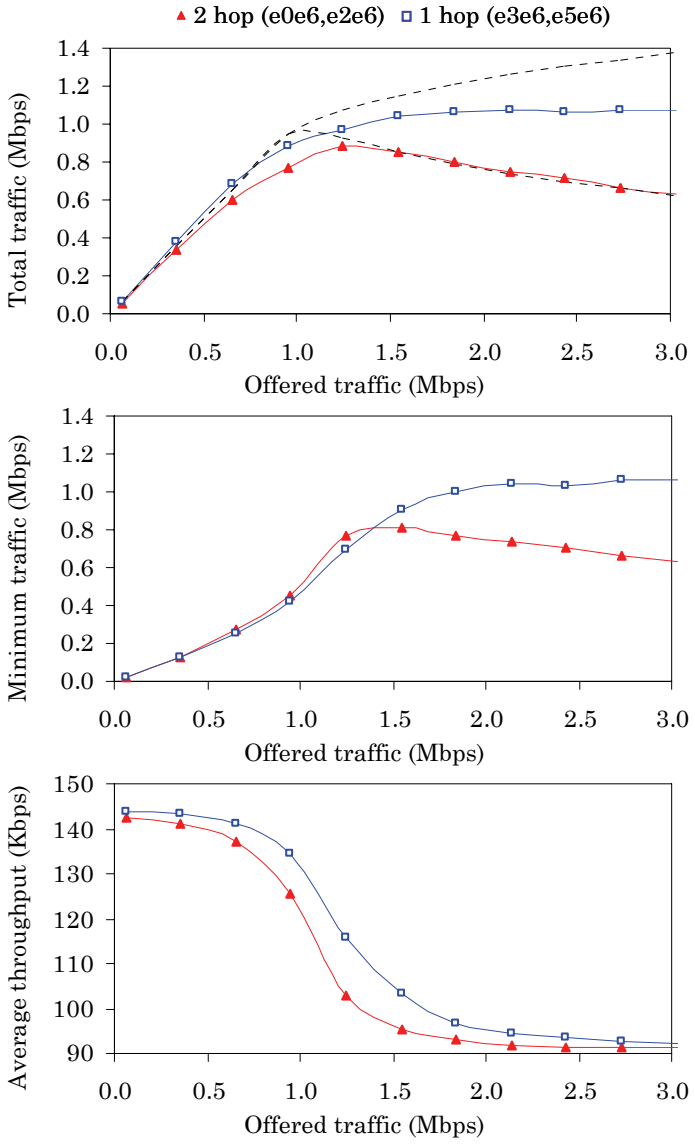


FIGURE 5-14: The influence of the number of hops on the performance of our 1st scheme with the modified TCP, in Topology 4 (Tfl_{06} = Tfl_{26} = 0.4 s, Tfl_{36} = Tfl_{56} = 0.2 s).

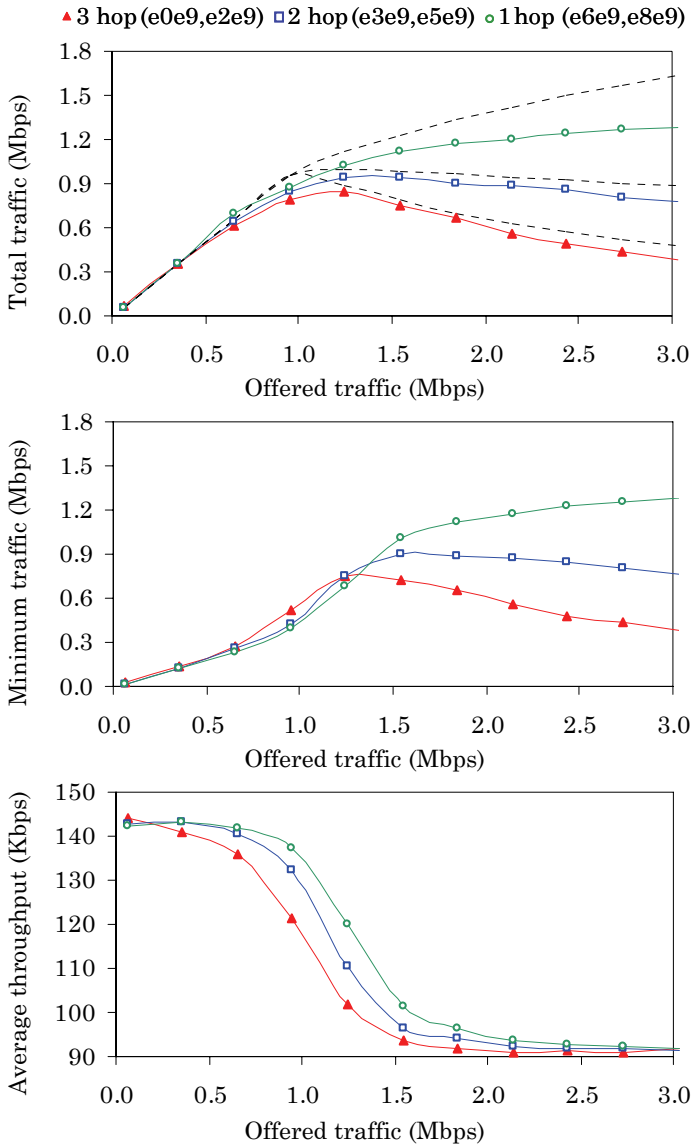


FIGURE 5-15: The influence of the number of hops on the performance of our 1st scheme with the modified TCP, in Topology 5 (Tfl_{09} = Tfl_{29} = 0.5 s, Tfl_{39} = Tfl_{59} = 0.4 s, Tfl_{69} = Tfl_{89} = 0.2 s).

5.4 Conclusions

In this chapter we presented our 1st scheme with AC for TCP elastic traffic. It considers that a flow is a sequence of related packets within a TCP connection, and it is able to provide different minimum throughputs (to flows from different users) and isolation between flows.

The scheme is built with simple mechanisms without using per-flow state, per flow signaling or per-flow processing in the network core. The whole scheme uses a small set of packet classes, each with a different discarding priority. The AC is implicit, edge-to-edge and based on per-flow throughput measurements. The first packets of the flow are used as a probing flow.

Flow's throughput is measured at the egress, and then signaling packets carry this measurement to the ingress, where the AC decision is made.

The scheme requires using a special modification of TCP sources. The short-term fluctuations of the sending rate of a "standard" TCP source reduce the performance, and in order to avoid this situation, we have proposed a modification of TCP's sending algorithms that keeps the short-term sending rate close to the actual average and above a minimum value.

The scheme considers a multidomain scenario in which each domain has a user-provider agreement (SLAs) with each of its neighboring domains. We have discussed the interdomain aspects and explained the details of the edge node operations.

We have evaluated the performance of our 1st scheme with the modified TCP through simulation. We have studied the influence of certain parameters in several network topologies using different traffic loads consisting of modified TCP flows that carry files of varying sizes. The results confirm that our 1st scheme with the modified TCP guarantees the requested minimum throughput to the accepted flows and achieves good utilization of the network resources. We have shown that it provides the desired minimum throughput to a lot more flows than the "traditional" scheme (best-effort service). We have also shown that it is able to provide different minimum throughput (to flows from different users) and isolation between flows. We have studied the influence of the measurement duration, and shown that the scheme obtains better performance using a short measurement duration, although there is a limit, because if it is too short, the measurements are wrong. We have also shown that the scheme discriminates against the multi-hop flows, but this is a similar behavior to the one observed in other AC schemes.

Chapter 6:

The 2nd scheme: a network scheme for TCP elastic traffic with AC using edge-to-edge per-aggregate measurements in class-based networks

In this chapter we describe our second proposal of a network scheme with AC for TCP elastic traffic. The main goal is to achieve a better performance for "standard" TCP flows since our 1st scheme requires a special modification of TCP's sending algorithms. The scheme considers that a flow is a sequence of related packets within a TCP connection, and it is able to provide different minimum throughputs (to flows from different users) and isolation between flows. It considers a multidomain scenario in which each domain has a user-provider agreement (SLA) with each of its neighboring domains. It is built with simple mechanisms without using per-flow state, per flow signaling or per-flow processing in the core. The whole scheme uses a small set of packet classes, specifically, three classes plus the best-effort class, with a different discarding priority assigned to each one. The AC is implicit, edge-to-edge and based on per-aggregate throughput measurements. The *out* packets of the aggregation of accepted flows, and in second term, a special flow (with packets marked as the highest discarding priority class), are used together as a probing flow to test if the throughput in the network path is enough to satisfy the flow's request. Then signaling packets carry the per-aggregate throughput measurement from the egress to the ingress, where the AC decision is made. Through simulation we show that our 2nd scheme with the "standard" TCP guarantees the requested minimum throughput to accepted flows and achieves a good utilization of resources.

The chapter is organized as follows. First, we present the basic features of the scheme and we discuss the reasons behind its architecture. Then we describe the scheme in detail. After that we show the simulation results we obtained to evaluate the performance. Finally we present the conclusions.

6.1 Introduction

In the previous chapter we showed that our 1st scheme guarantees the MTS to TCP flows, but it requires using a special modification of the sending rate algorithms of TCP sources since its

performance gets worse when using "standard" TCP sources. Therefore, the main goal of our 2nd scheme is to achieve a better performance for "standard" TCP flows than our 1st scheme, although it has the same general requirements and its architecture also has the same basic features (see Sections 1.2 and 5.1):

– It should guarantee the MTS to the maximum possible number of flows, where a flow is defined as a sequence of related packets within a TCP connection. It should be able to provide different minimum throughputs to different users. It should provide isolation between flows. It should be built with simple mechanisms, i.e., with mechanisms (of traffic conditioning, queue disciplines and AC) that reduce the per-flow state, per-flow signaling and per-flow processing as much as possible. It should consider a multidomain scenario in which each domain has a user-provider agreement (SLA) with each of its neighboring domains.
– It uses packet classes. The AC is edge-to-edge, based on measurements and implicit. The SLS part of the user-provider agreement defines the value of the maximum aggregated throughput the user may ask for, as well as the values of the port numbers and/or the specific mark in the packets that indicates that a flow requests the MTS and the desired minimum throughput. It uses pre-established LPs from ingress points to egress points.

In our 1st scheme we showed that the short-term fluctuations of the sending rate of a "standard" TCP source, above and below the average, cause some inaccuracies in the measurements that reduce the performance (see Subsection 5.2.2 and Figure 5-4 in Subsection 5.3.3). The throughput of each single flow is measured at the egress during a given time period. For a "standard" TCP flow, the measurement duration should be long in order to average its short-term rate fluctuations and obtain a correct measurement. However, as we also showed (Figure 5-8 in Subsection 5.3.3), if the measurement duration is too long, the performance in overloading gets worse. This is because long measurement durations imply that many flows meet simultaneously during the AC phase competing for the same resources, the per-flow measurement reflects an equal sharing of the resources, the AC tends to reject too many flows and the utilization decreases.

Unlike our 1st scheme, the AC of our 2nd scheme has some similarities mainly with the architecture of active measurement-based edge-to-edge AC schemes with per-aggregate probing, and also with the non-intrusive nature of passive measurement-based edge-to-edge AC schemes (Subsections 3.5.2 and 3.5.3): in the first ones, a single probing flow, related to an aggregation of accepted flows, is generated and sent continuously through the path, and its QoS is measured to be used in the AC decision; in the second ones, the egress node measures the QoS of the aggregation of accepted flows continuously to use it in the AC decision, and there is no intrusive traffic.

Unlike the AC of our 1st scheme, which was based on per-flow probing using the first packets of the flow, the AC of our 2nd scheme is based on per-aggregate probing: it uses mainly the *out* packets of the aggregation of accepted flows (the AOUT class), and in second term, it uses an extra probing flow (with packets marked as the highest discarding priority class) that is sent continuously from ingress to egress through the path. Note that although the extra probing flow is extra traffic, it is not intrusive because of the chosen marking, and that measurements are mainly passive and in second term they are active. The per-aggregated throughput measurement is sent periodically in a signaling packet from egress to ingress, where it is used to obtain an estimation of the available throughput in the path. In this way, instead of using per-flow measurements as in our 1st scheme, we use per-aggregate measurements, which will reduce the impact of the short-term fluctuations of TCP flows. Moreover, egress-to-ingress signaling is reduced. Another consequence of the chosen architecture is that the AC decision is fast as it is made as soon as a new flow arrives at the ingress, since the available throughput

in a path is known in advance. Another difference between the two schemes is that there are three classes here instead of four (plus the best-effort class).

Finally, note that, as in our 1st scheme, the throughput guarantee is qualitative (low loss if the sending rate is not higher than the minimum throughput) because of the measurements, and that the extra throughput comes from the best-effort sharing of the remaining resources between flows.

6.2 Description of the scheme

In this section we describe our 2nd scheme in detail. First, we deal with the main parts of the architecture, i.e., the definition of the packet classes, how the AC works and the edge-to-edge per-aggregate measurements. Then we deal with the interdomain aspects and the operations performed by edge nodes, which are very similar to the ones for our 1st scheme.

6.2.1 The architecture

When the first packet of a new flow *f* arrives to the network, the flow is assigned to a logical path *LP*, the list of active flows at the ingress is updated, and the requested minimum throughput of the flow $Rreq_{LP}^{f}$ (in [bps]) is obtained from the corresponding user-provider agreement. The AC evaluates whether this minimum throughput requirement can be provided without losing the minimum throughput guaranteed to the accepted flows. If the flow is accepted, it receives the MTS, and if the flow is rejected, it receives the best-effort service.

Our 2nd scheme (Figure 6-1) uses four packet classes: two A (Acceptance) classes, A_{IN} and A_{OUT}, the PR (Probing) class and the BE (Best-Effort) class. In the output links of all routers there is a FIFO queue (to maintain packet ordering) with discarding priorities for each class in the order (from low to high) A_{IN}, A_{OUT}, BE and PR.

The AC decision is made as soon as the first packet of the new flow arrives at the ingress, since the actual available throughput in the path *LP*, $Rav_{LP}(t)$ (in [bps]), is known in advance at the ingress. The flow is accepted if

$$Rreq_{LP}^{f} \leq Rav_{LP}(t), \tag{6-1}$$

and otherwise is rejected. If it is accepted, it receives the MTS, and its packets are marked as A: specifically, depending on the comparison between the average sending rate and the desired minimum throughput, each flow's packet is classified as in-profile or out-profile, and then *in* packets are marked as A_{IN} and *out* packets as A_{OUT}. If the flow is rejected, it receives the best-effort service, and its packets are marked as BE.

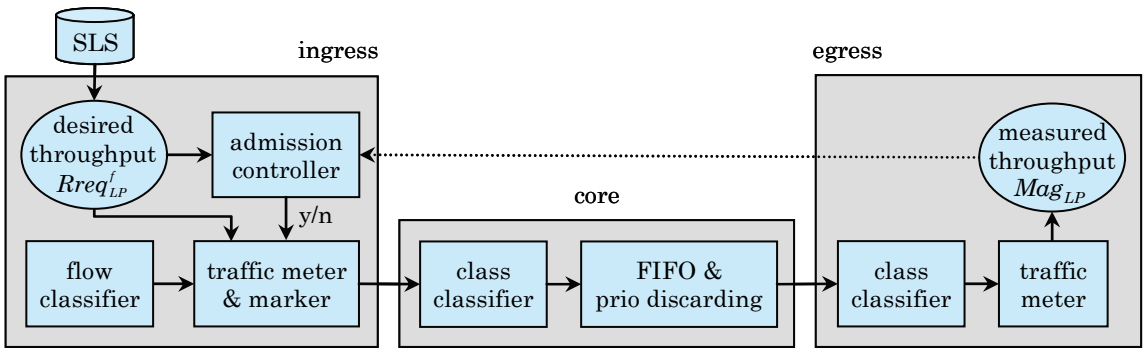


FIGURE 6-1: Functional block diagram of our 2nd scheme.

In order to discover the available throughput in the logical path Rav_{LP}, we perform aggregated measurements over some of the classes received at the egress. To help in the measurement we use a special flow that is sent continuously through the logical path. It is a CBR flow with packets marked as PR. The egress node periodically measures the aggregated throughput of AOUT, BE and PR classes (i.e., all classes except AIN) of flows assigned to the logical path. The obtained per-aggregate measurement Mag_{LP} (in [bps]) is an estimation of Rav_{LP}, although it requires some corrections, as we explain below. The value of Mag_{LP} is sent periodically to the ingress through a signaling packet.

Note that the available (unreserved) throughput in a link is the portion of the link's capacity that is not used by the traffic AIN of all the logical paths passing through this link. This unreserved link's capacity is used by the rest of the traffic according to the discarding priorities of classes AOUT, BE and PR (the lowest discarding priority class achieves the resources first, the next class the remaining ones, etc.), and proportionally to the input traffic load of each class and each logical path. This means that the available throughput is distributed between the logical paths that coincide in the link. The distribution is carried out in each link of the logical path until reaching the egress. Therefore, the obtained measurement Mag_{LP} is an estimation of the available throughput for the new arriving flows assigned to the logical path. Note that if the aggregated load of classes AOUT, BE and PR did not entirely fill the available throughput in the link, then the measurement would be pessimistic. However, this can be solved by increasing the rate of the special flow marked as PR. This is the reason why we have introduced this special flow in the architecture.

The measurement process of the aggregated throughput at the egress is the following. A class classifier selects the packets marked as AOUT, BE and PR corresponding to a logical path *LP* (see Figure 6-1). The measurements are made in time periods of duration Tag_{LP} (in [s]). Then Mag_{LP} is simply the total number of bits of these received packets during the measurement period, Bag_{LP} (in [bit]), divided by Tag_{LP}, i.e.,

$$Mag_{LP} = \frac{Bag_{LP}}{Tag_{LP}} \quad [\text{bps}]\,. \tag{6-2}$$

At the end of each measurement period a signaling packet is sent to the ingress carrying Mag_{LP}. Note that this egress-to-ingress signaling for each logical path is much less than the one required by our 1st scheme, where a signaling packet is sent for each new flow.

The value of Rav_{LP} is not directly the measurement Mag_{LP} since some corrections are required. Once a flow is accepted, the measurement does not immediately take into account the resulting decrease in the available throughput. Some time is needed to obtain an updated measurement at the egress, and to carry the updated measurement to the ingress. Specifically, in a given time t, where Mag_{LP} is the last measurement of a logical path *LP* received at the ingress at time t_I ($t_I \le t < t_I+Tag_{LP}$), we consider that the following accepted flows are not taken into account in Mag_{LP} (Figure 6-2):

- Accepted flows in *LP* during the time interval [t_I-rtt_{IE}-Tag_{LP}, t_I-rtt_{IE}], where rtt_{IE} is the ingress-egress RTT, because they are measured in Mag_{LP} during too short a time.
- Accepted flows in *LP* during [t_I-rtt_{IE}, t], because they are not measured in Mag_{LP}.

Note that we only need to take into account the accepted flows in *LP* and not the ones accepted in other coincident logical paths, since Mag_{LP} is an estimation of the available throughput only for this *LP*. To obtain Rav_{LP} we follow a conservative approach: we simply subtract from Mag_{LP} the desired minimum throughput of the accepted flows in *LP* not taken into account in Mag_{LP}; on the contrary, when the accepted flows end, we do not modify Mag_{LP} explicitly, but we allow the measurement to adapt to the traffic changes. We use the resulting value as the new Rav_{LP} for future AC decisions. Summing up, the available throughput in the path *LP* in the time t is

$$Rav_{LP}(t) = Mag_{LP} - \sum_{f} Racc_{LP}^{f} \ , \tag{6-3}$$

where Mag_{LP} is the last received measurement, and $Racc_{LP}^{f}$ (in [bps]) is the minimum throughput of a flow f that has been accepted during $[t_I\text{-}rtt_{IE}\text{-}Tag_{LP}, t]$.

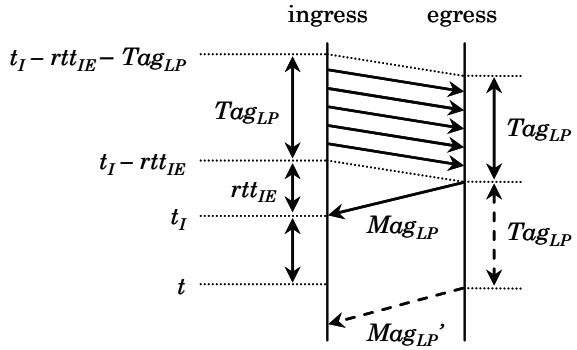


FIGURE 6-2: Obtaining the available throughput in a path Rav_{LP} from the measurement Mag_{LP}.

6.2.2 Interdomain aspects and edge node operations

The interdomain aspects for our 2nd scheme are the same as for our 1st scheme, while the edge node operations mainly differ for the operations at the egress. Therefore, the following is a short description and more details can be found in Subsection 5.2.4 of the 1st scheme.

Figure 6-3 shows a basic scenario with an upstream domain U, our domain O (the one using our 2nd scheme) and the downstream domain D. For interconnecting our domain O, when acting as a user (and D as a provider) or a provider (and U as a user), we use the same implicit method as in our 1st scheme: the start of a flow is detected at the domain ingress when its first packet is received; the end of a flow is detected when no packet is received within some defined timeout interval; the request of the MTS and the desired minimum throughput is indicated through the port numbers and/or a specific mark in the packets, in a way specified in the user-provider agreement; acceptance is indicated simply by providing the MTS, while rejection by providing the best-effort service.

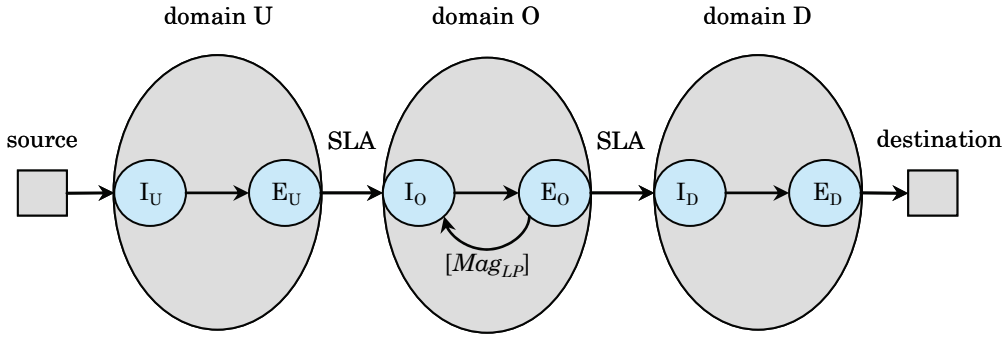


FIGURE 6-3: Interdomain operation of a domain using our 2nd scheme with neighboring domains.

The operations performed at the ingress of our domain are the following:

– Classifying packets into flows, implicitly detecting the start and end of flows and updating the list of active flows.

– Assigning new flows to logical paths.
– Obtaining the desired flow's throughput from the agreement with the upstream domain.
– Checking whether both the new request and the currently accepted ones are within the contracted value specified in the agreement.
– Making the AC decision (based on receiving signaling from the egress carrying the per-aggregate throughput measurement).
– Measuring the corresponding flow's traffic and marking after the AC decision.
– Registering services (accepted and rejected flows) provided to the upstream domain.

The following operations are performed at the egress of our domain:

– Measuring the aggregated throughput.
– Periodically sending signaling packets to the corresponding ingress for each logical path (carrying the per-aggregate throughput measurement).

6.3 Evaluation of the scheme

In this section we evaluate the performance of our 2nd scheme. Firstly, we study its scalability in terms of the number of flows passing through a router. Then we evaluate the performance through simulations as we did for the 1st scheme (Subsection 5.3). We also implement the different parts of the scheme in the ns-2 simulator [ns-2] and we use the same statistical traffic models and performance parameters. We describe these different aspects of the simulation again in the second subsection. In the third subsection we present the simulation results, which have similar goals and structure as in our 1st scheme. We study the influence of some parameters on the performance in several network topologies using different traffic loads consisting of ("standard") TCP flows that carry files of varying sizes. We show the results for the throughput obtained by the flows as well as for the efficiency in using the links' capacity.

6.3.1 Scalability study

We analyze the use of network resources (memory and processor in routers, and link's capacity) by the operations performed by different algorithms in edge and core routers, in terms of N, the number of flows passing through a router. This scalability study is very similar to the one carried out for our 1st scheme (Subsection 5.3.1), except for the operations performed at the egress.

The operations performed by edge routers at the ingress are the following (Subsection 6.2.2): 1) classifying packets into flows, implicitly detecting the start and end of flows and updating the list of active flows; 2) assigning new flows to logical paths and obtaining the desired flow's throughput from the agreement; 3) the AC decision, after receiving signaling packets from the egress carrying the throughput measurement; 4) the measurement of the input traffic of the flow and the in/out marking after the AC decision. The processing time of all these operations is constant with respect to N except in the first one, the lookup in the list of active flows, which can be implemented with algorithms O(log N), considered as scalable in [coul05a]. The use of memory in all of these operations is constant with respect to N except in the first one, which is O(N), considered as scalable in [coul05a]. Using link's capacity is not applicable in any of these operations.

The operations performed by edge routers at the egress are the following (Subsection 5.2.4): 1) measuring the aggregated throughput; 2) sending the signaling packet (one per logical path) periodically to the corresponding ingress with the throughput measurement. The processing time and the use of memory of all these operations is constant with respect to N, while using

link's capacity is not applicable in the first one and is constant with respect to N in the second one.

Finally, the operations performed in core routers are: 1) classifying each packet into a class; 2) packet scheduling and forwarding based on FIFO and packet discarding based on priorities per class. The processing time and the use of memory in all of these operations is constant with respect to N, while using link's capacity is not applicable in the first one and is constant with respect to N in the second on e.

6.3.2 Description of the simulation

The description of the simulation for our 2nd scheme is very similar to the one for our 1st scheme (Subsection 5.3.2). Here we provide a summary and only go into detail for those aspects that are different.

Implementation of the scheme in the simulator

We have implemented the functional blocks of our 2nd scheme (Subsection 6.2.1) within the Diffserv module of the ns-2 simulator [ns-2]. At the ingress routers, for each arriving flow, there is a sending rate meter, a marker and an admission controller, and at the egress routers, for each logical path, there is a throughput meter. In the output links of all routers there is a FIFO queue with priority discarding according to packet class. The CBR flow of packets marked as PR in each logical path is generated using tools that are already available in the simulator (a sender, a traffic meter and marker based on the token bucket and a receiver). Specifically, we have implemented the following:

- As in our 1st scheme, the FIFO queue with priority discarding for the four packet classes (in the order, from low to high, A_{IN}, A_{OUT}, PR and BE), which drops a packet with the highest discarding priority when there is no available buffer space.
- The throughput meter periodically counts the total received bytes during the measurement duration of classes A_{OUT}, BE and PR from a given logical path, and then it notifies the ingress router of the throughput measurement Mag_{LP} (with the corresponding packet delay).
- As in our 1st scheme, for the "standard" TCP, the sending rate meter is based on the TSW algorithm and the marker is probabilistic (see Subsection 4.1.2). The TSW is characterized by the *win_lenght* parameter (we use the flow's RTT) and the marker is characterized by the average rate's threshold (we use the desired minimum throughput). The marker here uses three marks (instead of five) to identify the three packet classes A_{IN}, A_{OUT} and BE: packets are marked as A_{IN} or A_{OUT} if the flow is accepted, and as BE if it is rejected.
- The admission controller computes the available throughput in the path Rav_{LP} from the received measurement Mag_{LP}, then it compares Rav_{LP} with the desired flow's minimum throughput $Rreq^{f}_{LP}$ and performs the AC decision.
- As in our 1st scheme, all TCP sources consider the users' impatience, that is, the source stops sending if the transfer time is too high.

Appendix A provides more details about the implementation in the ns-2 simulator.

Network topologies and statistical traffic model

We use the same Topologies 1, 2, 3, 4 and 5, as in the evaluation of our 1st scheme (see Figure 5-3 in Subsection 5.3.2). We use a "standard" TCP source, specifically TCP NewReno. We generate TCP flows that carry a single file from an ingress point to an egress point through a logical path. The statistical traffic model is the same as in our 1st scheme, i.e., each flow is

characterized by the file size, obtained from a Pareto distribution, and the starting time, which is obtained from a Poisson arrival process. Using all these statistical distributions, the average offered traffic load from an ingress point to an egress point through a logical path is equal to $\lambda\sigma$ bps, where λ (in [flow/s]) is the average number of arrivals per second and σ (in [bit]) is the average file size.

For each logical path we generate the same quantity of offered traffic load, which we vary from 0.05 Mbps to 3 Mbps in steps of 0.3 Mbps, in order to study underloading and overloading situations in the links. In the simulations we usually choose 90 Kbps as the value of the desired minimum throughput for all TCP flows (which means, for a document of 10 packets, around 10 Kbytes, a minimum transfer time of around 0.9 s) and the user's impatience is twice the desired file transfer time (which means, for 90 Kbps of desired throughput, getting a throughput of approximately 45 Kbps).

Performance parameters

We define the same performance parameters as in our 1st scheme, i.e., we consider the satisfied flows (the ones that complete the file transfer and get at least 95% of the desired minimum throughput, e.g., 85.5 Kbps for 90 Kbps), and then we obtain, for each logical path, the three performance parameters: the average "total" satisfied traffic load, the average "minimum" satisfied traffic load, and the average throughput of satisfied flows (which will always be a value above 95% of the desired minimum throughput). These parameters are obtained in the same way as for our 1st scheme, i.e., from the throughput values of each satisfied flow, filtering the packets of the aggregation of satisfied flows, and calculating the time averages without considering the initial period of the simulation transient phase. The simulation length was chosen so that after this initial transient phase a minimum of 10,000 flows is generated. We made 10 independent replications of each simulation, and estimated the mean value by computing the sample mean and the 95% confidence interval [law00a].

6.3.3 Simulation results

In this subsection we present the simulation results we have obtained for evaluating the performance of our 2nd scheme. We study the influence of some parameters and we show results for the throughput obtained by the flows as well as for the efficiency in using the links' capacity. Specifically, the different "cases" we present are the following:

- 1st case (Figure 6-4). We compare our 2nd scheme, our 1st scheme and the "traditional" scheme (best-effort service).
- 2nd case (Figure 6-5). We analyze in detail the throughput provided by our 2nd scheme to verify that it guarantees the requested minimum throughput to the accepted flows.
- 3rd case (Figure 6-6). We study the influence of the measurement duration Tag_{LP} on the performance of our 2nd scheme.
- 4th case (Figure 6-7). We study the ability of our 2nd scheme to provide different minimum throughputs to flows from different users.
- 5th case (Figure 6-8). We study the ability of our 2nd scheme to provide isolation between flows.
- 6th case (Figure 6-9). We study the influence of the packet RTT on the performance of our 2nd scheme.
- 7th case (Figure 6-10). We study the fairness in the sharing of resources between logical paths achieved by our 2nd scheme.
- 8th case (Figures 6-11, 6-12 and 6-13). We study the influence of the number of hops of the logical paths on the performance of our 2nd scheme.

In the 1st case we compare our 2nd scheme, our 1st scheme and the "traditional" scheme (best-effort service). The results are shown in Figure 6-4. We use TCP NewReno, a desired minimum throughput for all TCP flows of 90 Kbps, and Topology 1, with measurement durations Tag_{03} = Tag_{23} = 0.5 s for our 2nd scheme and Tfl_{03} = Tfl_{23} = 0.5 s for our 1st scheme. In the figure we show the results for the satisfied traffic of the "bottleneck" link, i.e., the sum of the two values of the total satisfied traffic of each logical path, the sum of the two values of the minimum satisfied traffic of each logical path and the average between the two values of the average throughput of satisfied flows of each logical path, all three versus the offered traffic load of each logical path, from 0 to 3 Mbps (therefore the offered traffic load to the link is the sum of the offered traffic load of each logical path, i.e., from 0 to 6 Mbps). Note that, ideally, if the maximum utilization were achieved, the total and the minimum satisfied traffic load would be similar and around the bottleneck link's capacity of 2 Mbps and the average throughput of satisfied flows would be around the desired minimum throughput of 90 Kbps.

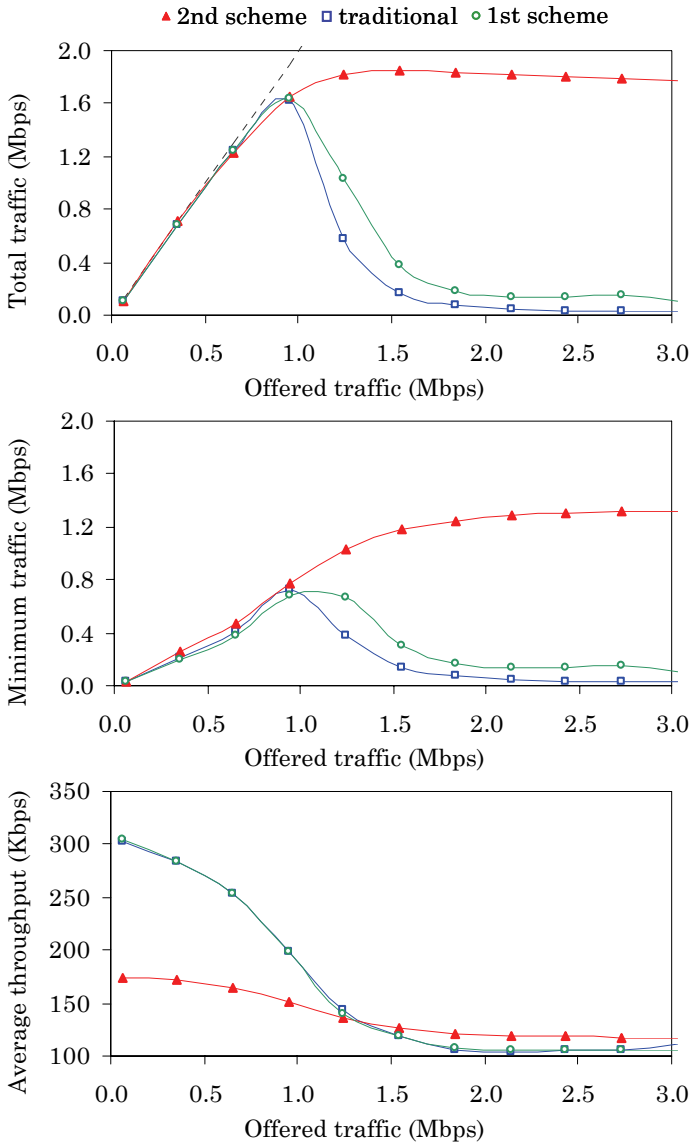


FIGURE 6-4: Comparison of our 2nd scheme, our 1st scheme and the "traditional" scheme (best-effort service), using TCP NewReno in Topology 1 (Tag_{03} = Tag_{23} = 0.5 s, Tfl_{03} = Tfl_{23} = 0.5 s).

The results in Figure 6-4 prove that our 2nd scheme is much better than the "traditional" scheme and also better than our 1st scheme when "standard" TCP flows are used. In underloading, when there are enough resources to satisfy all flows, all schemes achieve the same value for the total satisfied traffic load, which at the same time is equal to the offered traffic load (the dashed line indicates the equality). In overloading, the utilization of the "traditional" scheme tends to zero, similarly to our 1st scheme (see Subsections 5.2.2 and 5.3.3), while the utilization of our 2nd scheme is high and stays almost constant for the entire range of offered traffic loads (the total satisfied traffic load is around 1.8 Mbps and the minimum satisfied traffic load is about 1.3 Mbps). Moreover, as expected, the average throughput of satisfied flows decreases for high values of the offered traffic load because the unreserved resources decrease (the final value for our 2nd scheme is about 117 Kbps).

In the 2nd case, we analyze in detail the throughput provided by our 2nd scheme to verify that it guarantees the requested minimum throughput to the accepted flows. Some of the results obtained are shown in Figure 6-5. We use TCP NewReno, a desired minimum throughput for all TCP flows of 90 Kbps and Topology 1, with measurement durations $Tag_{03} = Tag_{23} = 0.5$ s. From the simulation results we obtain the percentage of accepted flows that complete the file transfer, the percentage of accepted flows that are satisfied (the ones that complete the file transfer and get at least the threshold throughput: 95% of the desired minimum throughput, i.e., 85.5 Kbps), and the frequency distribution of the throughput of the accepted flows to determine how far the accepted and non-satisfied flows are from the threshold value. From this analysis we conclude the following: all accepted flows complete the file transfer; a high percentage of accepted flows are satisfied; a high percentage of the accepted and non-satisfied flows achieve a throughput close to the threshold throughput.

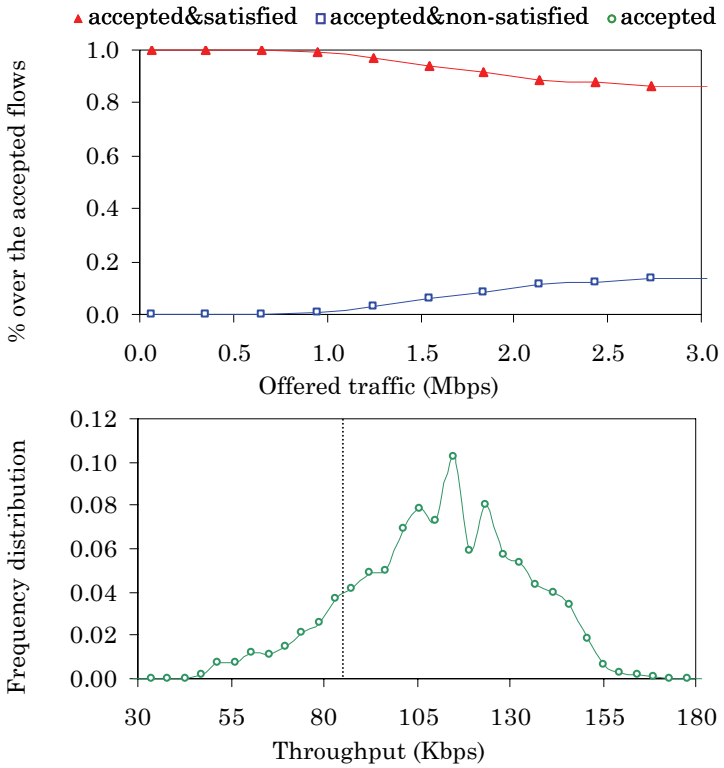


FIGURE 6-5: For our 2nd scheme and TCP NewReno in Topology 1 ($Tag_{03} = Tag_{23} = 0.5$ s), at the top, percentage of satisfied flows and non-satisfied flows over the accepted flows versus the offered traffic load, and at the bottom, frequency distribution of throughput of accepted flows when the offered traffic load of each logical path is 3 Mbps (in total, 6 Mbps).

The graph at the top of Figure 6-5 shows percentages over the accepted flows versus the offered traffic load of each logical path. The percentage of accepted flows that are satisfied is 100% for a large range of offered traffic loads and it decreases slowly for high values. The graph at the bottom shows the frequency distribution of throughput of accepted flows (the dashed line indicates the threshold throughput) when the offered traffic load is 3 Mbps (in total, 6 Mbps), i.e., in the worst case. Although about 13% of flows are not satisfied, a throughput smaller than 63 Kbps is only achieved by 3% of flows, which is a very small value. Also note the following: although the utilization of our 2nd scheme with TCP NewReno is generally lower than the utilization of our 1st scheme with the modified TCP (Subsection 5.3.3), these results shows that the conservative approach we use for obtaining Rav_{LP} from Mag_{LP} (Subsection 6.2.1) is necessary, since if we tried to increase the number of accepted flows (i.e., the utilization), the percentage of accepted flows that are satisfied would decrease.

In the 3rd case, we study the influence of the measurement duration Tag_{LP} on the performance of our 2nd scheme. The results are shown in Figure 6-6.

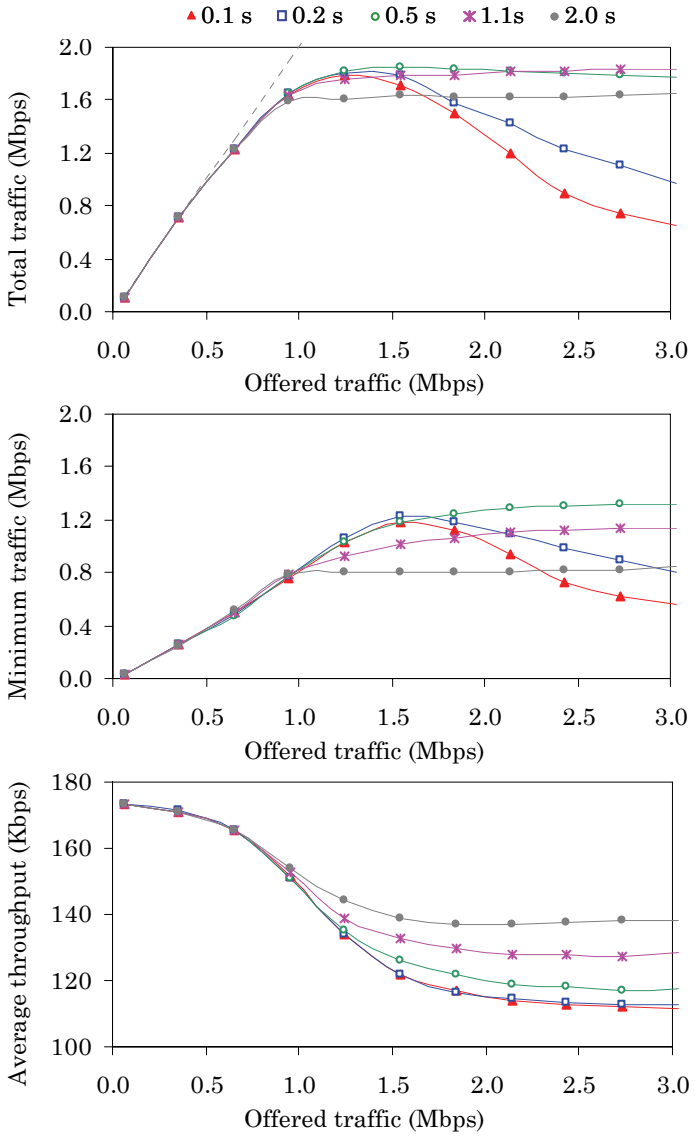


FIGURE 6-6: The influence of the measurement duration Tag_{LP} on the performance of our 2nd scheme and TCP NewReno, in Topology 1 ($Tag_{03} = Tag_{23}$).

We use TCP NewReno, a desired minimum throughput for all TCP flows of 90 Kbps, and Topology 1 with equal measurement duration for both logical paths $Tag_{03} = Tag_{23}$. In the figure we show the three performance parameters for the satisfied traffic of the "bottleneck" link versus the offered traffic load of each logical path. We use five different measurement durations, 0.1 s, 0.2 s, 0.5 s, 1.1 s and 2.0 s (note that the packet inter-arrival time for a CBR flow with a packet length of 1,000 bytes and a rate of 90 Kbps would be about 0.09 s). The measurement duration of 0.5 s achieves the best performance, and when Tag_{LP} is shorter (0.2 s and 0.1 s) or longer (1.1 s and 2.0 s), the performance of the 2nd scheme gets worse (the utilization decreases).

The reason for the behavior of the curves in Figure 6-6 is the following. As in the 1st scheme (see Subsection 5.3.3), it is related to the measurement process, but the reason is different since in the 1st scheme we used per-flow measurements while here we are using per-aggregate measurements. Note that although the number of flows does not change, the traffic load presents fluctuations in the sort-term and the measurements reflect this. When Tag_{LP} is too short, the measurement is more sensitive to bursts in the traffic load and therefore the performance gets worse. A longer Tag_{LP} reduces the effect of these short-term fluctuations, but if it is too long it slows down the measurement's reaction to the departure and arrival of flows, and again the performance gets worse. This behavior is similar to the one observed in classical measurement-based hop-by-hop AC schemes (see Subsection 3.3.2). In conclusion, the performance of our 2nd scheme gets worse when the measurement duration Tag_{LP} is too short or too long.

In the 4th case, we study the ability of our 2nd scheme to provide different minimum throughputs to flows from different users. The results are shown in Figure 6-7. We use TCP NewReno and Topology 1, with measurement durations $Tag_{03} = Tag_{23} = 0.5$ s. We consider two users, A and B, with desired minimum throughput of 90 and 135 Kbps respectively. User A sends flows through the logical path e0e3 and user B through e2e3. For each user the offered traffic load is the same and varies from 0 to 3 Mbps. In the figure we show the three performance parameters for each user (and also for all together) versus the offered traffic load of each user. The results in Figure 6-7 show that our 2nd scheme achieves the desired different average throughputs (the final values are about 181 and 114 Kbps respectively), which is a consequence of the differentiation between the *in* and *out* traffic of the flow. Moreover, there are some differences in the sharing of the link between the two users (similarly to the 1st scheme, although there the differences were even smaller than here, see Subsection 5.3.3). These differences would mean that the AC accepts more flows demanding a small minimum throughput than flows demanding a large one. However, this is a similar behavior to the one observed in other AC schemes where flows demanding more resources experience a higher blocking probability (see, e.g., Subsections 3.3 and 3.5.1).

In the 5th case, we study the ability of our 2nd scheme to provide isolation between flows, so that flows sending more traffic than their allocated throughput do not damage well-behaved flows that do send according to their allocated throughput. The results are shown in Figure 6-8. We use TCP NewReno and Topology 1, with measurement durations $Tag_{03} = Tag_{23} = 0.5$ s. We send TCP flows together with CBR flows. The desired minimum throughput of all flows (TCP and CBR) is 90 Kbps, but CBR flows have a constant rate of 135 Kbps while TCP flows adjust the sending rate as usual. CBR flows are generated in a similar way as TCP flows: the starting times are obtained from a Poisson arrival process (characterized by λ flow/s, the average number of arrivals per second), and the time duration is Fs/R, where R is the flow's sending rate (135 Kbps), and Fs is a "file size" (in [bit]) obtained from a Pareto distribution with the same parameters used for the TCP flows. As a consequence, the offered traffic load of the set of CBR flows is again $\lambda\sigma$ bps. We send TCP flows through the logical path e0e3 and CBR flows through e2e3. For each type of flows the offered traffic load is the same and varies

from 0 to 3 Mbps. In the figure we show the three performance parameters for each type of flow (and also for all together) versus the offered traffic load of each type of flow.

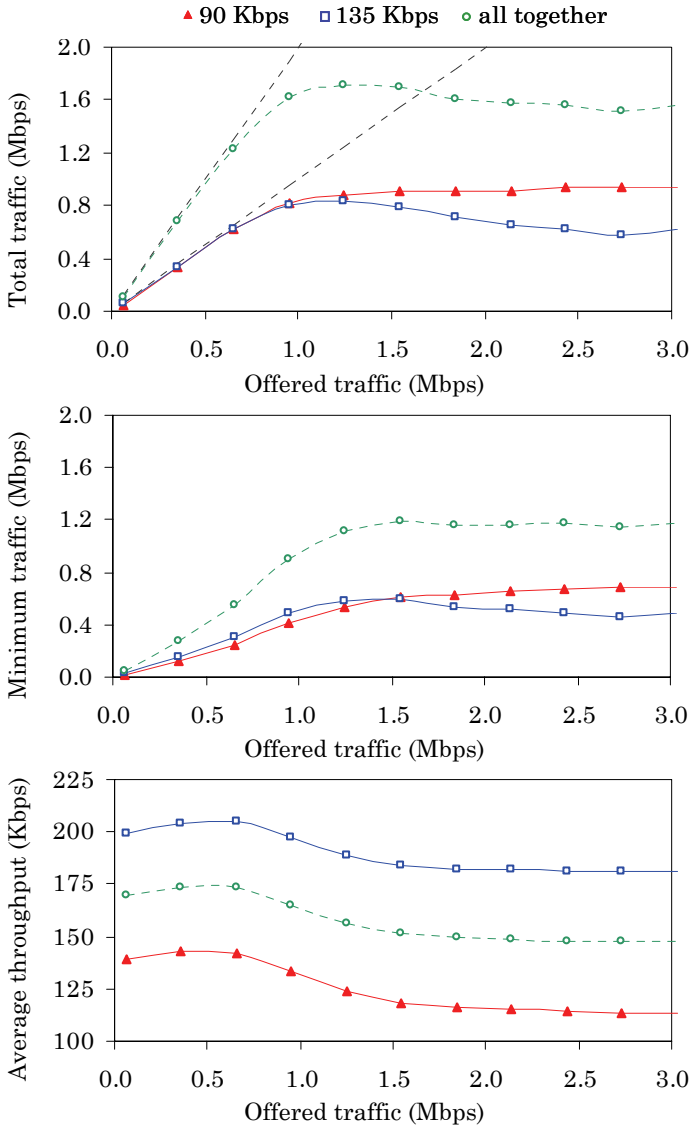


FIGURE 6-7: For our 2nd scheme and TCP NewReno, results for two users asking for different minimum throughputs, 90 Kbps for user A and 135 Kbps for user B, in Topology 1 ($Tag_{03} = Tag_{23} =$ 0.5 s).

The results in Figure 6-8 show that the accepted TCP and CBR flows achieve the desired minimum throughput even though CBR flows are sending constantly at a higher rate. Again this is because our 2nd scheme differentiates between the *in* and *out* traffic of the flow. Moreover, there are some differences in the sharing of the link between the two types of flows as well as in the extra throughput. This is because the unreserved resources are only shared fairly between CBR flows because they do not adjust the sending rate like TCP, but have a maximum per-flow throughput of 135 Kbps; the rest is shared fairly between TCP flows.

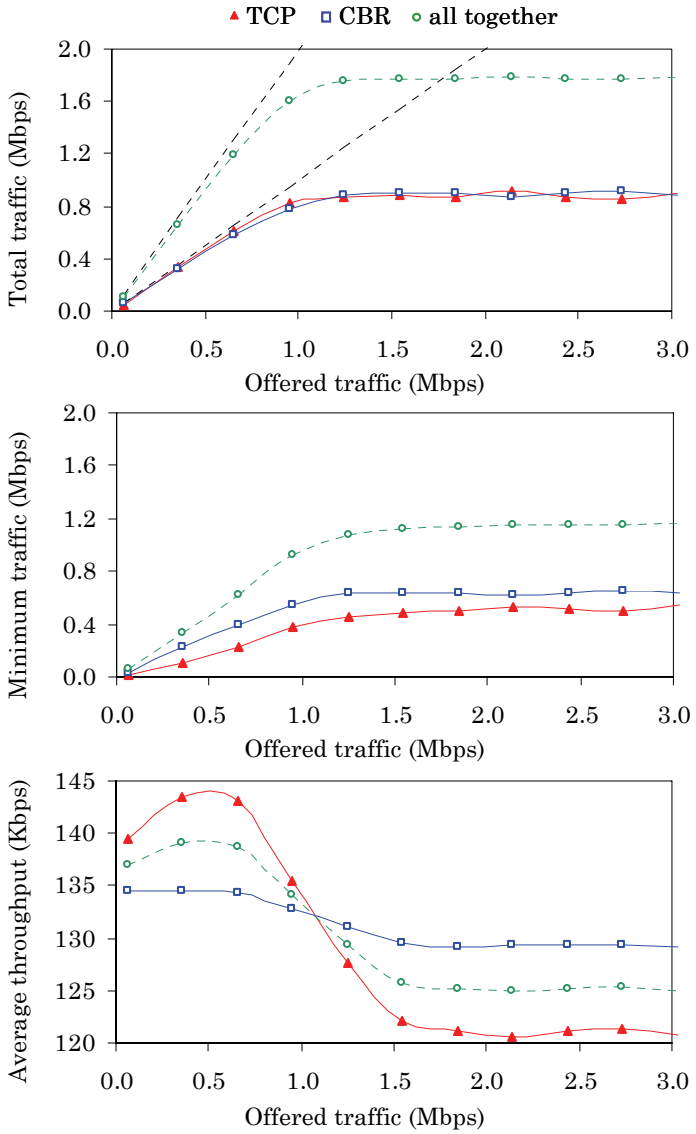


FIGURE 6-8: For our 2nd scheme and TCP NewReno, isolation of TCP flows against CBR flows of 135 Kbps, when the desired minimum throughput of TCP and CBR flows is 90 Kbps, in Topology 1 ($Tag_{03} = Tag_{23} = 0.5$ s).

In the 6th case, we study the influence of the packet RTT on the performance of our 2nd scheme. The results are shown in Figure 6-9. We use the TCP NewReno, a desired minimum throughput for all TCP flows of 90 Kbps, and Topology 2, where the packet RTT for the logical path e0e3 is larger than the one for the logical path e2e3 (specifically, the RTT without taking into account the queuing time is, for e0e3, around 165 ms for hosts and 84 ms for edge routers, and for e2e3, 89 ms and 49 ms respectively; the maximum queuing time is 200 ms). The measurement durations are $Tag_{03} = Tag_{23} = 0.5$ s. In the figure we show the three performance parameters for each logical path (and also for all together) versus the offered traffic load of each logical path. The results show that both logical paths achieve similar utilization and therefore the performance does not depend on the packet RTT. Also note that the average throughput of satisfied flows is different for both logical paths, which is a consequence of the traditional RTT unfairness of TCP (see Subsection 2.4.3).

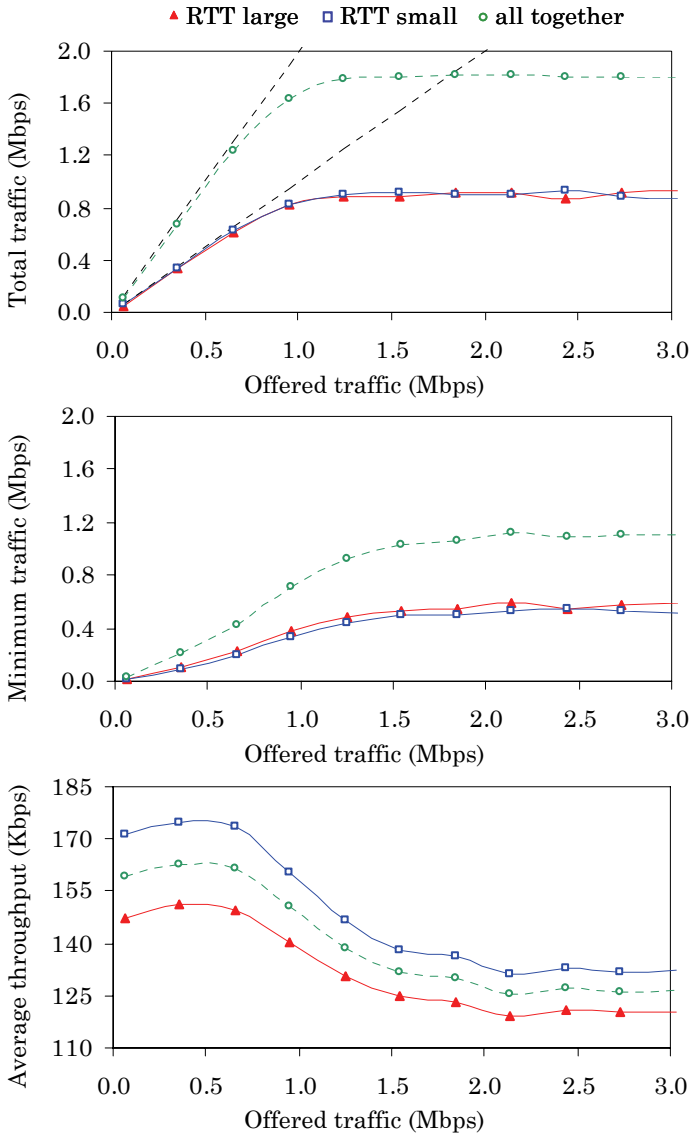


FIGURE 6-9: The influence of the packet RTT on the performance of our 2nd scheme and TCP NewReno, in Topology 2 (Tag_{03} = Tag_{23} = 0.5 s).

In the 7th case, we study the fairness in the sharing of resources between logical paths. The results are shown in Figure 6-10. We use the TCP NewReno, Topology 1, measurement durations Tag_{03} = Tag_{23} = 0.5 s and a desired minimum throughput for all TCP flows of 90 Kbps. We keep the offered traffic load of the logical path e0e3 constant at 1 Mbps and we vary the offered traffic load of the logical path e2e3 from 0 to 3 Mbps. In the figure we show the three performance parameters for each logical path versus the offered traffic load of the logical path e2e3. We expect that the 2 Mbps of the bottleneck link will be shared between the two logical paths proportionally to their respective offered traffic load (this expected result is indicated by the dashed lines). Note that the simulation results are quite close to the expected results.

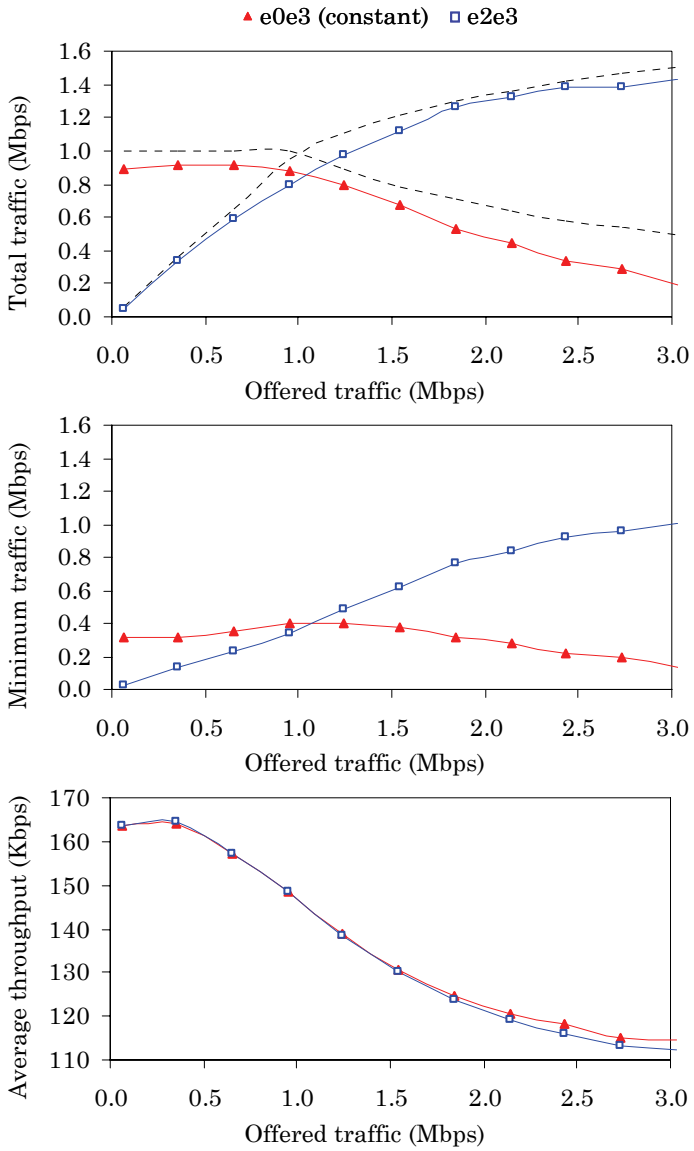


FIGURE 6-10: For our 2nd scheme and TCP NewReno, sharing between logical paths in Topology 1 (Tag_{03} = Tag_{23} = 0.5 s). The offered traffic load of e0e3 is constant and equal to 1 Mbps.

In the 8th case, we study the influence of the number of hops of the logical paths on the performance of our 2nd scheme. We use three different topologies, 3, 4 and 5, and we show the results in Figures 6-11, 6-12 and 6-13 respectively. We use TCP NewReno, a desired minimum throughput of 90 Kbps and the same measurement duration Tag_{LP} = 0.5 s for all logical paths. In the figures, for clarity, we do not show the results for each logical path (since the number of logical paths is large), but we combine the results of the logical paths with the same number of hops into a single result: we show, for the three performance parameters, the average values between the pair of logical paths with the same number of hops (1, 2 or 3 hops), all three versus the offered traffic load of each logical path (except in Figure 6-11 for the logical path e0e8, because it is the only one with 2 hops in Topology 3).

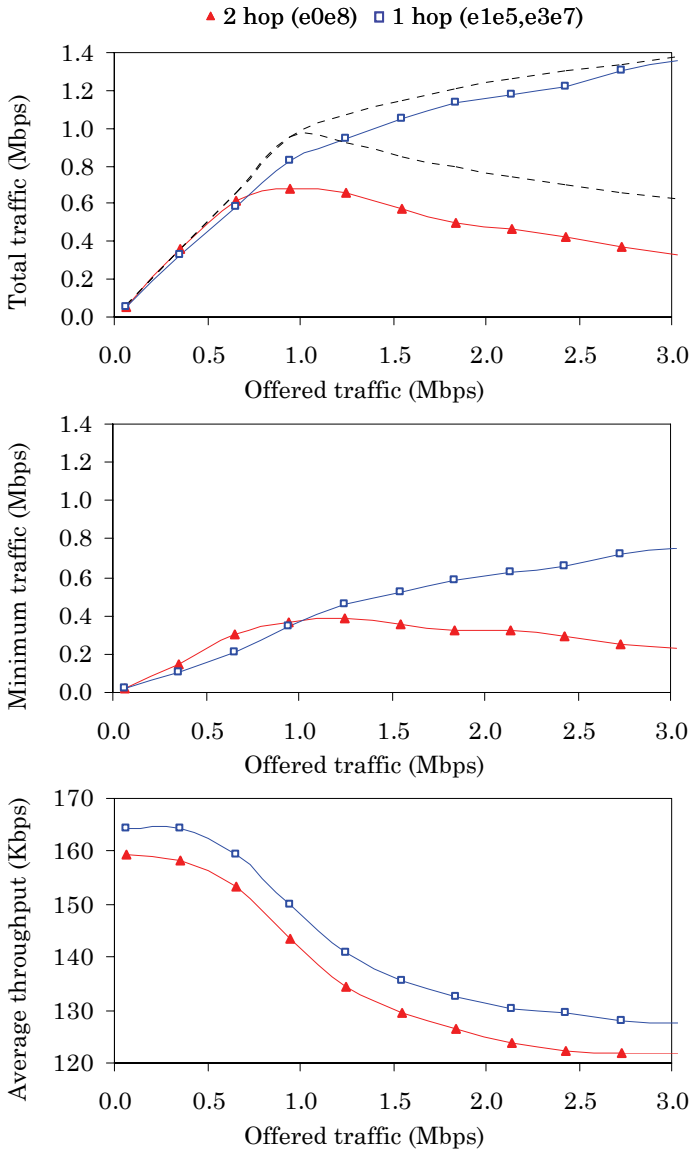


FIGURE 6-11: The influence of the number of hops on the performance of our 2nd scheme and TCP NewReno, in Topology 3 ($Tag_{08} = Tag_{15} = Tag_{37} = 0.5$ s).

The results in Figures 6-11, 6-12 and 6-13 show that in congestion, the logical paths with fewer hops obtain greater utilization than the logical paths with more hops. Logical paths with multiple hops obtain a smaller value of satisfied traffic load since flows passing through multiple congested links experience a higher blocking probability. However, this is a similar behavior to the one observed in other AC schemes (see, e.g., Subsections 3.3 and 3.5.1). In order to check this similarity, we have compared the results obtained by our 2nd scheme with the ideal results that would be obtained by a classical hop-by-hop AC scheme. We obtain this theoretical result taking into account that in our topologies the blocking probability in each core link is the same and that the offered traffic load for each logical path is also the same. These ideal results are represented by dashed lines in the curves of the total satisfied traffic load. Note that the simulation results for the three topologies follow the ideal results, although the total utilization is lower.

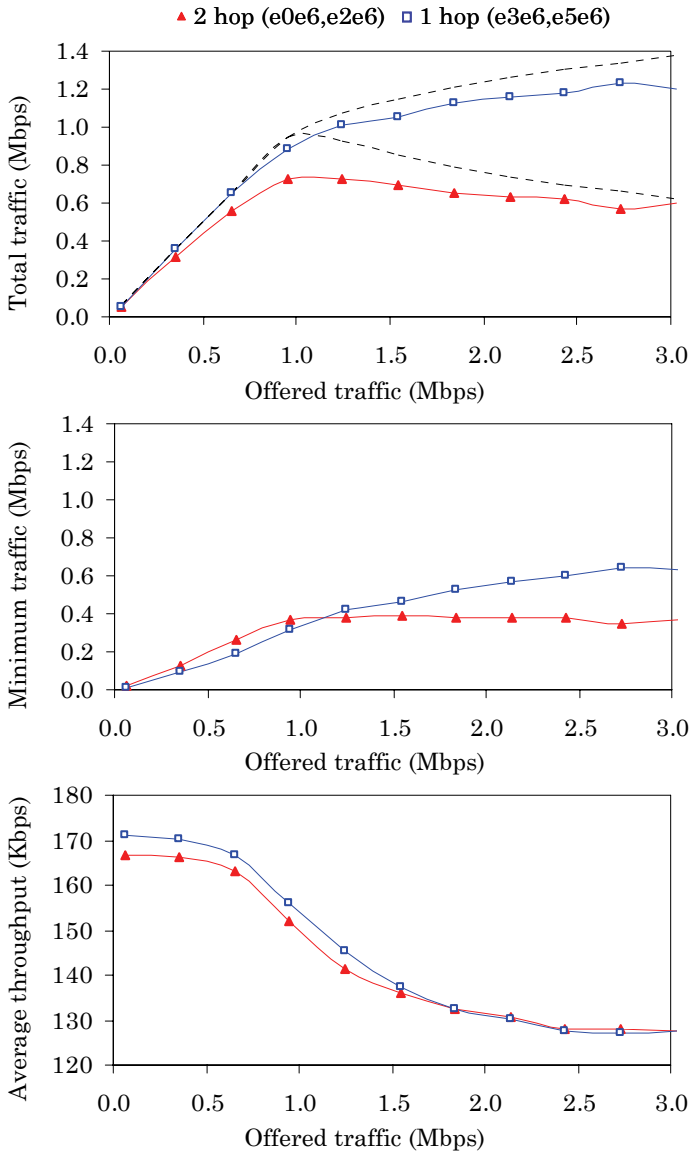


FIGURE 6-12: The influence of the number of hops on the performance of our 2nd scheme and TCP NewReno, in Topology 4 ($Tag_{06} = Tag_{26} = Tag_{36} = Tag_{56} = 0.5$ s).

6.4 Conclusions

In this chapter we presented our 2nd scheme with AC for TCP elastic traffic. The main goal was to achieve a better performance for "standard" TCP flows since our 1st scheme requires a special modification of TCP's sending rate algorithms. The scheme considers that a flow is a sequence of related packets within a TCP connection, and it is able to provide different minimum throughputs (to flows from different users) and isolation between flows.

The scheme is built with simple mechanisms without using per-flow state, per flow signaling or per-flow processing in the network core. The whole scheme uses a small set of packet classes, each with a different discarding priority. The AC is implicit, edge-to-edge and based on

per-aggregate throughput measurements. The *out* packets of the aggregation of accepted flows, and in second term, a special flow (with packets marked as the highest discarding priority class), are used together as a probing flow. The aggregated throughput is measured at the egress, and then signaling packets carry this measurement to the ingress, where the AC decision is made.

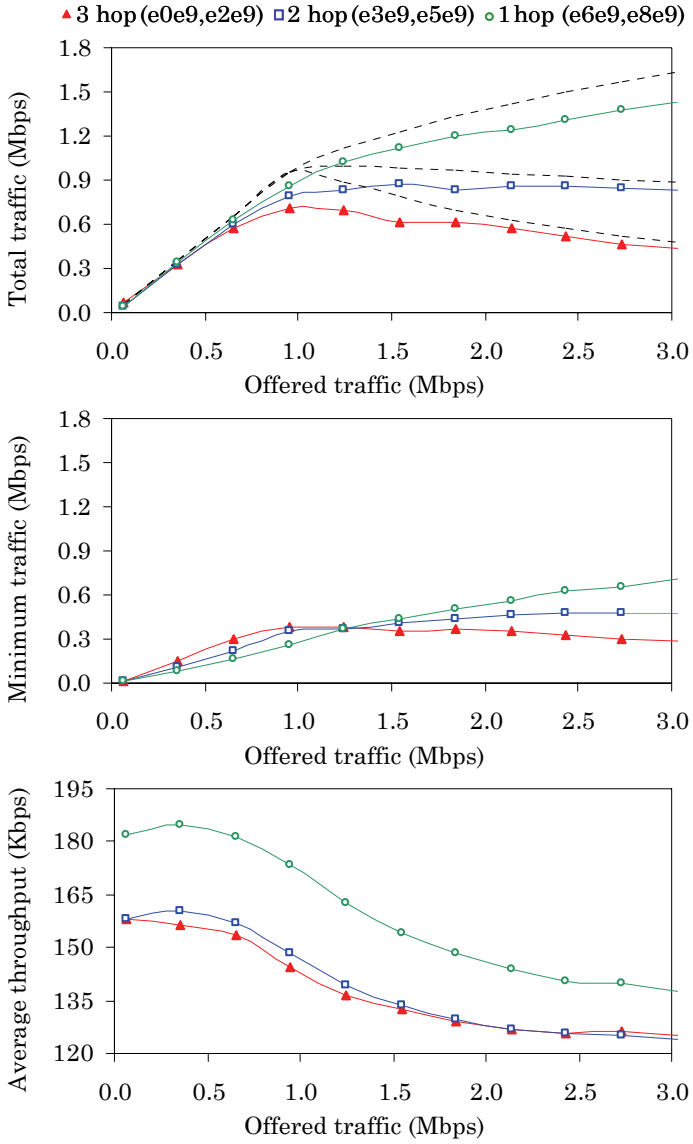


FIGURE 6-13: The influence of the number of hops on the performance of our 2nd scheme and TCP NewReno, in Topology 5 ($Tag_{09} = Tag_{29} = Tag_{39} = Tag_{59} = Tag_{69} = Tag_{89} = 0.5$ s).

The scheme considers a multidomain scenario in which each domain has a user-provider agreement (SLAs) with each of its neighboring domains. We have discussed the interdomain aspects and explained the details of the edge node operations.

We have evaluated the performance of our 2nd scheme with "standard" TCPs (TCP NewReno) through simulations. We have studied the influence of some parameters in several network topologies using different traffic loads consisting of TCP flows that carry files of varying sizes. The results confirm that our 2nd scheme guarantees the requested minimum throughput to

the accepted flows and achieves good utilization of the network resources. We have shown that it provides the desired minimum throughput to a lot more flows than the "traditional" scheme (best-effort service). We have also shown that it is able to provide different minimum throughputs (to flows from different users) and isolation between flows. We have studied the influence of the measurement duration, and shown that if it is too short or too long, the performance gets worse. We have also shown that the scheme discriminates against multi-hop flows, but this is a similar behavior to the one observed in other AC schemes.

Chapter 7:

Conclusions and future work

In this chapter we summarize the main contributions of this work and point out possible directions for future research.

7.1 Conclusions

Our main goal was to design a network scheme with AC for TCP elastic traffic using simple mechanisms, and more specifically, a network scheme with the following requirements: it should guarantee the MTS to the maximum possible number of flows, where a flow is defined as a sequence of related packets within a TCP connection; it should be able to provide different minimum throughputs to different users and isolation between flows; it should be built with simple mechanisms, i.e., with mechanisms (of traffic conditioning, queue disciplines and AC) that reduce the per-flow state, per-flow signaling and per-flow processing as much as possible; it should consider a multidomain scenario in which each domain has a user-provider agreement (SLA) with each of its neighboring domains.

In order to design the desired network scheme, we have performed the following studies:

– We have studied the basic concepts of multiservice network architectures, the definition and building of services through network mechanisms (especially the role of AC) and the characteristics of TCP elastic traffic (the network service it requires and several aspects of TCP).
– We have studied the main AC schemes that have been proposed for the Internet, focusing on their simplicity regarding how many nodes participate in the AC, the required state, the use of signaling, and others. Using implicit ways of communication between nodes and traffic measurements (active or passive) in the AC algorithms can considerably reduce the number of nodes that are aware of AC, the state they maintain and the required signaling. In this direction, edge-to-edge AC schemes are a good option, since only edge nodes participate in the AC decision and maintain a per-flow state, while core nodes do not maintain either a per-flow or aggregate reservation state and do not require signaling. This feature is the main advantage in relation to other AC schemes: in hop-by-hop AC schemes, each node maintains the actual aggregated reservation and performs a local AC decision, and per-flow

signaling and/or per-flow state are required; in one-hop AC schemes on LPs with resource reservation, the AC decision is only at the ingress node, but per-path resource reservations require more complex management to avoid the LPs passing through the same link share resources in a non-appropriate way.

– We have studied the main network schemes that have been proposed in the Internet for TCP elastic traffic, with or without AC, focusing on the main characteristics of the service (whether the minimum throughput can be different or is the same for all flows, whether isolation between flows is provided, etc.) and its architecture (the specific traffic conditioning, queue disciplines and AC mechanisms used, the required state, the use of signaling, etc.). Out of the different network schemes without AC, the ones based on packet classes show a good trade-off between simplicity (per-flow operations are kept just at the edge) and the service characteristics (they allow different throughputs and isolation between flows to be provided). However, these network schemes do not use AC. Out of the network schemes with AC we have studied, we found that is of special interest the definition of flow used in the scheme for elastic traffic in Cross-Protect, as it captures the sequences of packets that occur as bursts within persistent TCP connections, as well as the implicit way of detecting the start and end of these flows. Another interesting aspect of some of these schemes with AC is the utilization of implicit ways for indicating the requested service parameters (QoS and traffic), although they achieve this by providing the same minimum throughput to all flows. In all of them the AC is hop-by-hop and based on measurements. However, they require either per-flow signaling in the core, or are not able to provide different throughputs or isolation between flows, or require per-flow state and per-flow queuing in the core.

We have proposed two network schemes and evaluated their performance through simulations. The two schemes have a similar architecture but use different kinds of AC:

– Both network schemes are based on packet classes, with a different discarding priority assigned to each one. The AC is edge-to-edge and based on measurements, so that only edge routers participate in the AC, exchange signaling and maintain a list of active flows, which is used to detect new flows and to isolate the accepted flows by enforcing the agreed traffic profile on their input traffic. The AC is implicit: the start of a new flow is detected at the ingress when its first packet is received; the end of a flow is detected when no packet is received within a defined timeout interval; the request of the MTS and the desired minimum throughput are indicated through the port numbers and/or a specific mark in the packets, in the way specified in the user-provider agreement; acceptance is indicated simply by providing the MTS, while rejection by providing the best-effort service. The user-provider agreement also defines the value of the maximum aggregated throughput the user may ask for. Finally, the schemes also use pre-established LPs from ingress points to egress points. With these common basic features, both network schemes are simple since per-flow state, per flow signaling and per-flow processing are not required in the network core.
– In our 1st scheme the AC is based on per-flow throughput measurements. The first packets of the flow are used as a probing flow. Flow's throughput is measured at the egress, and then signaling packets carry this measurement to the ingress, where the AC decision is made. The scheme requires using a special modification of TCP sources. The short-term fluctuations of the sending rate of a "standard" TCP source reduce the performance, and to avoid this situation, we have proposed a modification of TCP's sending algorithms that keeps the short-term sending rate close to the actual average and above a minimum value. We have evaluated the performance of our 1st scheme with the modified TCP through simulations. We have studied the influence of certain parameters on several network topologies using different traffic loads consisting of modified TCP flows that carry files of varying sizes. The results confirm that our 1st scheme with the modified TCP guarantees the requested minimum throughput to the accepted flows and achieves good utilization of

the network resources. We have shown that it provides the desired minimum throughput to a lot more flows than the "traditional" scheme (best-effort service). We have also shown that it is able to provide different minimum throughputs (to flows from different users) and isolation between flows. We have studied the influence of the measurement duration, and shown that the scheme obtains better performance using a short measurement duration, although there is a limit, because if it is too short, the measurements are wrong. We have also shown that it discriminates against the multi-hop flows, but this is a similar behavior to the one observed in other AC schemes.

– Our 2nd scheme aimed to achieve a better performance for "standard" TCP sources than our 1st scheme. The AC is based on per-aggregate throughput measurements. The *out* packets of the aggregation of accepted flows, and in second term, a special flow (with packets marked as the highest discarding priority class), are used together as a probing flow. The aggregated throughput is measured at the egress, and signaling packets carry this measurement to the ingress, where the AC decision is made. We have evaluated the performance of our 2nd scheme with "standard" TCPs (TCP NewReno) through simulation. We have studied the influence of some parameters on several network topologies using different traffic loads consisting of TCP flows that carry files of varying sizes. The results confirm that our 2nd scheme guarantees the requested minimum throughput to the accepted flows and achieves good utilization of the network resources. We have shown that it provides the desired minimum throughput to a lot more flows than the "traditional" scheme (best-effort service). We have also shown that it is able to provide different minimum throughput (to flows from different users) and isolation between flows. We have studied the influence of the measurement duration, and shown that if it is too short or too long, the performance gets worse. We have also shown that it discriminates against multi-hop flows, but this is a similar behavior to the one observed in other AC schemes.

Finally, the main results of this research work have been published in several international journals and presented in different international conferences. Appendix B provides a detailed list of the related publications.

7.2 Future work

There are several topics in which the actual research work can be continued in the future. The following list contains some topics related to improving or extending particular aspects of the current work and other broader topics that point to future research directions:

– Evaluating the performance of the schemes in more complex simulation scenarios. Firstly, although we have already considered several network topologies, it is also important to obtain results in bigger and more realistic topologies. Secondly, we have generated the TCP traffic (at different loads) as an aggregation of single independent document transfers (with sizes according to a Pareto distribution, which approximates the heavy tail behavior observed in Internet traffic measurements reasonably well), but it would be interesting to generate it as an aggregation of several related document transfers within application sessions (web, peer-to-peer file sharing, ftp, e-mail…) and measure the provided throughput during the sessions. Thirdly, bidirectional (data) traffic has not been considered and studying the behavior of the schemes in these situations would also be interesting.
– Developing an analytical model for the optimal measurement duration. The simulation results have shown that the performance of both schemes depends on the measurement duration. We have used manual tuning to find the best values (the ones with the highest resource utilization) and therefore an analytical model would be useful for predicting them in advance. It would also provide more insight into the measurement process of each

scheme. The optimal measurement duration is expected to depend on traffic characteristics such as flow durations and network topology parameters.

– Studying the fairness between short and long flows and developing a traffic conditioner that takes into consideration flow durations. In the "traditional" scheme short flows tend to achieve smaller throughput than long flows, especially because they spend most of their lifetime in Slow Start and are more sensitive to losses. We expect that our schemes accept and satisfy flows regardless of their duration since the *in* packets of accepted flows are protected, but a further study would be necessary to confirm this. Moreover, this study should include the extra throughput in order to determine if the remaining resources are shared equally between short and long flows. The results of this study could motivate development of a new traffic conditioner (traffic meter and marker) for TCP flows that takes into consideration flow durations and compensates for the possible unfairness. On the other hand, as we have already mentioned (see Subsection 2.4.3), another way of sharing the remaining resources between flows for providing the extra throughput is giving preference to short flows over long flows. This could also motivate the development of a new traffic conditioner that takes flow durations into consideration.
– Modifying the AC algorithms in order to avoid multirate and multihop unfairness. The simulation results have shown that a flow demanding a larger minimum throughput or passing through more congested links is more likely to be rejected, a behavior that is similar to that observed in other AC schemes. Exploring different modifications of the AC algorithm so that it not only takes into account if there are enough resources to satisfy the flow but also the range of possible values of requested minimum throughputs (see the "trunk reservation" mechanism in [cost242]) and number of congested links could be a way to reach a solution in the future.
– Extending the proposed schemes for TCP elastic traffic to include real-time traffic. The goal of this future research would be to design a network scheme with AC that provides a minimum throughput service to elastic flows and a low-jitter, no loss service to real-time flows, using a similar architecture. It would use packet classes (scheduling priority classes for the real-time flows and discarding priority classes for the elastic flows), the corresponding queue discipline (with scheduling and discarding priorities), and a joint AC that is implicit, edge-to-edge and based on measurements.

Bibliography

[agga00a] "Understanding the performance of TCP pacing", A. Aggarwal, S. Savage, T. Anderson, Proceedings of the IEEE Infocom, 2000.

[alip05a] "XML Service Level Specification and validation", P. Alipio, S. Lima, P. Carvalho, Proceedings of the IEEE International Symposium on Computer Communications (ISCC), 2005.

[alme98a] "SRP: a scalable resource reservation protocol for the Internet", W. Almesberger, T. Ferrari, J.L. Boudec, Elsevier Computer Communications, vol. 21, no. 14, 1998.

[aras94a] "Real-time communication in packet-switched networks", C. Aras, J. Kurose, D. Reeves, H. Schulzrinne, Proceedings of the IEEE, vol. 82, no. 1, 1994.

[avra04a] "Differentiation between short and long TCP flows: predictability of the response time", K. Avrachenkov, U. Ayesta, P. Brown, E. Nyberg, Proceedings of the IEEE Infocom, 2004.

[awey02a] "A self-regulating TCP acknowledgment (ACK) pacing scheme", J. Aweya, M. Ouellette, D.Y. Montuno, International Journal of Network Management, vol. 12, no. 3 , 2002.

[bake97a] "Reservations about reservations", F. Baker, J. Crowcroft, R. Guerin, H. Schulzrinne, L. Zhang, Proceedings of the IFIP International Workshop on Quality of Service (IWQoS), 1997.

[bala98a] "TCP behavior of a busy Internet server: analysis and improvements", H. Balakrishnan, V.N. Padmanabhan, S. Seshan, M. Stemm, R.H. Katz, Proceedings of the IEEE Infocom, 1998.

[bans01a] "Analysis of SRPT scheduling: investigating unfairness", N. Bansal, M. Harchol-Balter, ACM SIGMETRICS Performance Evaluation Review, vol. 29, no. 1, 2001.

[ben01b] "Statistical bandwidth sharing: a study of congestion at flow level", S. Ben Fredj, T. Bonald, A. Proutière, G. Régnié, J.W. Roberts, ACM SIGCOMM Computer Communication Review, vol. 31, no. 4, 2001.

[bert87a] "Data networks", D. Bertsekas, R. Gallager, published by Prentice Hall, ISBN 0-132-00916-1, 1987.

[bhat00a] "QoS-sensitive flows: issues in IP packet handling", S. N. Bhatti, J. Crowcroft, IEEE Internet Computing, vol. 4, no. 4, 2000.

[bian00a] "Throughput analysis of end-to-end measurement-based admission control in IP", G. Bianchi, A. Capone, C. Petrioli, Proceedings of the IEEE Infocom, 2000.

[blum01a] "Rethinking the design of the Internet: the end-to-end arguments versus the brave new world", M.S. Blumenthal, D. D. Clark, ACM Transactions on Internet Technology, vol. 1, no 1, 2001.

[boch97a] "Some principles for quality of service management", G.V. Bochmann, A. Hafid, Distributed Systems Engineering, IOP Publishing, vol. 4, 1997.

[bona01d] "Insensitivity results in statistical bandwidth sharing", T. Bonald, A. Proutière, G. Régnié, J.W. Roberts, Proceedings of the International Teletraffic Congress (ITC), 2001.

[bona02b] "IP traffic and QoS control: the need for a flow-aware architecture", T. Bonald, S. Oueslati-Boulahia, J.W. Roberts, Proceedings of the World Telecommunications Congress (WTC), 2002.

[bona03b] "Congestion at flow level and the impact of user behavior", T. Bonald, J.W. Roberts, Elsevier Computer Networks, vol. 42, no. 4, 2003.

[borg99b] "VBR bandwidth guaranteed services over DiffServ networks", F. Borgonovo, A. Capone, L. Fratta, C. Petrioli, Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS), 1999.

[borg99c] "PCP: a bandwidth and delay guaranteed transport service for IP networks", F. Borgonovo, A. Capone, L. Fratta, M. Marchese, C. Petrioli, Proceedings of the IEEE International Conference on Communications (ICC), 1999.

[brak95a] "TCP Vegas: end to end congestion avoidance on a global Internet", L. Brakmo, L. Peterson, IEEE Journal on Selected Areas in Communication, vol. 13, no. 8, 1995.

[bres00a] "Comments on the performance of measurement-based admission control algorithms", L. Breslau, S. Jamin, S. Shenker, Proceedings of the IEEE Infocom, 2000.

[bres00b] "Endpoint admission control: architectural issues and performance", L. Breslau, E. Knightly, S. Shenker, I. Stoica, H. Zhang, ACM SIGCOMM Computer Communication Review, vol. 30, no. 4, 2000.

[bres98a] "Best-effort versus reservations: a simple comparative analysis", L. Breslau, S. Shenker, ACM SIGCOMM Computer Communication Review, vol. 28, no. 4, 1998.

[bres99a] "Measurement-based admission control: what is the research agenda?", L. Breslau, S. Jamin, S. Shenker, Proceedings of IEEE International Workshop on Quality of Service (IWQoS), 1999.

[brow02a] "Understanding Internet traffic streams: dragonflies and tortoises", N. Brownlee, KC Claffy, IEEE Communications Magazine, vol. 40, no. 10, 2002.

[card00a] "Modeling TCP latency", N. Cardwell, S. Savage, T. Anderson, Proceedings of the IEEE Infocom, 2000.

[case02a] "TCP Westwood: end-to-end congestion control for wired/wireless network", C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, R. Wang, Wireless Networks, vol. 8, no. 5, 2002.

[ceti01a] "Scalable services via egress admission control", C. Cetinkaya, V. Kanodia, E. W. Knightly, IEEE Transactions on Multimedia, vol. 3, no. 1, 2001.

[chen98a] "An overview of quality of service routing for next-generation high-speed networks: problems and solutions", S. Chen, K. Nahrstedt, IEEE Network, vol. 12, no. 6, 1998.

[chiu89a] "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks", D.M. Chiu, R. Jain, Computer Networks and ISDN Systems, vol. 17, no. 1, 1989.

[clar92a] "Supporting real-time applications in an integrated services packet network: architecture and mechanism", D. D. Clark, S. Shenker, L. Zhang, ACM SIGCOMM Computer Communication Review, vol. 22, no. 4, 1992.

[clar98a] "Explicit allocation of best-effort packet delivery service", D. D. Clark, W. Fang, IEEE/ACM Transactions on Networking, vol.6, no. 4, 1998.

[come95a] "Internetworking with TCP/IP. Volume I", D.E. Comer, published by Prentice Hall, ISBN 0-13-227836-7, 1995.

[cost242] "Broadband network teletraffic. Performance evaluation and design of broadband multiservice networks. Final report of Action COST 242", J.W. Roberts, U. Mocci, J. Virtamo (Eds.), Lecture Notes in Computer Science (LNCS), vol. 1155, published by Springer-Verlag, ISBN 3-540-61815-5, 1996.

[coul05a] "Distributed systems: concepts and design", G. Coulouris, J. Dollimore, T. Kindberg, published by Addison Wesley, ISBN 0-321-26354-5, 2005.

[crov97a] "Self-similarity in World Wide Web traffic: evidence and possible causes", M. E. Crovella, A. Bestavros, IEEE/ACM Transactions on Networking, vol. 5, no. 6, 1997.

[deme89a] "Analysis and simulation of a fair queuing algorithm", A. Demers, S. Keshav, S. Shenker, ACM SIGCOMM Computer Communication Review, vol. 19, no. 4, 1989.

[diffserv] IETF Differentiated Services (Diffserv) Working Group, http://www.ietf.org./html.charters/OLD/diffserv-charter.html.

[domi98a] "A framework for adaptive applications", J. Domingo-Pascual, J. Mangues-Bafalluy, Research report UPC-DAC-1998-7, Department of Computer Architecture (DAC) at the Technical University of Catalonia (UPC), 1998.

[eure99a] "A common framework for QoS/network performance in a multi-provider environment", Deliverable 1, The EQoS Framework, Eurescom Project P806-GI, 1999.

[fall96a] "Simulation-based comparisons of Tahoe, Reno, and SACK TCP", K. Fall, S. Floyd, ACM SIGCOMM Computer Communication Review, vol. 26, no. 3, 1996.

[feld00a] "Characteristics of TCP connection arrivals", A. Feldmann, Self-Similar Network Traffic and Performance Evaluation, K. Park, W. Willinger (Eds.), published by Wiley, ISBN 0-471-31974-0, 2000.

[feng99a] "Adaptive packet marking for maintaining end-to-end throughput in a differentiated-services Internet", W. Feng, D. D. Kandur, IEEE/ACM Transactions on Networking, vol. 7, no. 5, 1999.

[floyd] Sally Floy's homepage at ICIR (the Center for Internet Research at the International Computer Science Institute), http://www.icir.org/floyd/.

[floy92a] "On traffic phase effects in packet-switched gateways", S. Floyd, V. Jacobson, Internetworking: Research and Experience, vol. 3, no. 3, 1992.

[floy93a] "Random early detection gateways for congestion avoidance", S. Floyd, V. Jacobson, IEEE/ACM Transactions on Networking, vol. 1, no. 4, 1993.

[fral03a] "Packet-level traffic measurements from the Sprint IP backbone", C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, S.C. Diot, IEEE Network, vol. 17, no. 6, 2003.

[geor04a] "Provider-level Service Agreements for Inter-domain QoS delivery", P. Georgatsos, J. Spencer, D. Griffin, P. Damilatis, H. Asgari, J. Griem, G. Pavlou, P. Morand, Proceedings of the International Workshop on Advanced Internet Charging and QoS Technologies (ICQT), 2004.

[gros03a] "A time-scale decomposition approach to measurement-based admission control", M. Grossglauser, D.N.C. Tse, IEEE/ACM Transactions on Networking, vol. 11, no 4, 2003.

[guo01a] "The war between mice and elephants", L. Guo, I. Matta, Proceedings of the IEEE International Conference on Network Protocols (ICNP), 2001.

[intserv] IETF Integrated Services (Intserv) Working Group, http://www.ietf.org./html. charters/OLD/intserv-charter.html.

[ippm] IETF IP Performance Metrics (IPPM) Working Group, http://www.ietf.org/html. charters/ippm-charter.html.

[jaco88a] "Congestion avoidance and control", V. Jacobson, ACM SIGCOMM Computer Communication Review, vol. 18, no. 4, 1988.

[jaco98a] "Differentiated Services for the Internet", V. Jacobson, Proceedings of the Internet2 Joint Applications / Engineering QoS Workshop, 1998.

[jama03a] "Measurement-based admission control scheme with priority and service classes for application in wireless IP networks", A. Jamalipour, J. Kim, Wiley International Journal of Communication Systems, vol. 16, no. 6, 2003.

[jami97a] "A measurement-based admission control algorithm for integrated services packet networks", S. Jamin, P. B. Danzig, S. J. Shenker, L. Zhang, IEEE/ACM Transactions on Networking, vol. 5, no. 1, 1997.

[jami97b] "Comparison of measurement-based admission control algorithms for controlled-load service", S. Jamin, S. J. Shenker, P.B. Danzig, Proceedings of the IEEE Infocom, 1997.

[jin05a] "Fast TCP: from theory to experiments", C. Jin, D.X. Wei, S.H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh, IEEE Network, vol. 19, no. 1, 2005.

[kata02a] "Congestion control for high bandwidth-delay product networks", D. Katabi, M. Handley, C. Rohrs, ACM SIGCOMM Computer Communication Review, vol. 32 no. 4, 2002.

[kell03a] "Scalable TCP: improving performance in high speed wide area networks", T. Kelly, ACM SIGCOMM Computer Communication Review, vol. 32, no. 2, 2003.

[knig99a] "Admission control for statistical QoS: theory and practice", E. W. Knightly, N. B. Shroff, IEEE Network, vol. 13, no. 2, 1999.

[knoc97a] "Quantitative QoS-mapping: a unifying approach", H. Knoche, H. de Meer, Proceedings of the IFIP International Workshop on QoS, 1997.

[knoc98a] "QoS parameters: a comparative study for mapping purposes", H. Knoche, H. de Meer, Technical Report, University of Hamburg, 1998.

[kort04a] "Cross-protect: implicit service differentiation and admission control", A. Kortebi, S. Oueslati, J.W. Roberts, Proceedings of the IEEE Workshop on High Performance Switching and Routing (HPSR), 2004.

[kuma00a] "Nonintrusive TCP connection admission control for bandwidth management of an Internet access link", A. Kumar, M. Hegde, S.V.R. Anand, B.N. Bindu, D. Thirumurthy, A.A. Kherani, IEEE Communications Magazine, vol. 38, no. 5, 2000.

[kuma02a] "TCP-friendly traffic conditioning in DiffServ networks: a memory-based approach", K. R. R. Kumar, A. L. Ananda, L. Jacob, Elsevier Computer Networks, vol. 38, no. 6, 2002.

[kuro93a] "Open issue and challenges in proving quality of service guarantees in high-speed networks", J.F. Kurose, ACM SIGCOMM Computer Communication Review, vol. 23, no. 1, 1993.

[labr99a] "Packet dropping policies for ATM and IP networks", M. A. Labrador, S. Banerjee, IEEE Communications Surveys & Tutorials, vol. 2, no. 3, 1999.

[lai01b] "TCP congestion control algorithms and a performance comparison", Y. Lai, C. Yao, Proceedings of the IEEE International Conference on Computer Communications and Networks (ICCCN), 2001.

[law00a] "Simulation modeling and analysis", A. M. Law, W. D. Kelton, published by MacGraw-Hill, ISBN 0-07-059292-6, 2000.

[lieb99a] "Work-conserving vs. non-work-conserving packet scheduling: an issue revisited", J. Liebeherr, E. Yilmaz, Proceedings of the IEEE/IFIP International Workshop on Quality of Service (IWQoS), 1999.

[lima03a] "A distributed admission control model for CoS networks using QoS and SLS monitoring", S. Lima, P. Carvalho, A. Santos, V. Freitas, Proceedings of the IEEE International Conference on Communications (ICC), 2003.

[lima04a] "Measuring QoS in class-based IP networks using multi-purpose colored probing patterns", S. Lima, P. Carvalho, V. Freitas, Proceedings of SPIE ITCom – Performance, Quality of Service, and Control of Next-Generation Communication Networks, 2004.

[lima06a] "Distributed admission control in multiservice IP networks: concurrency issues", S. Lima, P. Carvalho, V. Freitas, Academy Publisher Journal of Communications (JCM), vol. 1, no. 3, 2006.

[marq01a] "Novel enhancements to load control - a soft-state, lightweight admission control protocol", Á. Marquetant, O. Pop, R. Szabó, G. Dinnyés, Z. Turányi, Proceedings of the 2nd COST263 International Workshop on Quality of Future Internet Services (QofIS), 2001.

[mass00a] "Bandwidth sharing and admission control for elastic traffic", L. Massoulié, J.W. Roberts, Telecommunication Systems, vol. 15, no. 1-2, 2000.

[mass02a] "Bandwidth sharing: objectives and algorithms", L. Massoulié, J.W. Roberts, IEEE Transactions on Networking, vol. 10, no. 3, 2002.

[mass99a] "Arguments in favor of admission control for TCP flows", L. Massoulié, J.W. Roberts, Proceedings of the International Teletraffic Congress (ITC), 1999.

[mort00a] "Implicit admission control", R. Mortier, I. Pratt, C. Clark, S. Crosby, IEEE Journal on Selected Areas in Communications, vol. 18, no. 12, 2000.

[mpls] IETF MultiProtocol Label Switching (MPLS) Working Group, http://www.ietf.org/html.charters/mpls-charter.html.

[mitz96a] "A study of reservation dynamics in integrated services packet networks", D. J. Mitzel, D. Estrin, S. Shenker, L. Zhang, Proceedings of the IEEE Infocom, 1996.

[nahr95a] "Resource management in networked multimedia systems", K. Nahrstedt, R. Steinmetz, IEEE Computer, vol. 28, no. 5, 1995.

[nich98a] "Using Diffserv Premium Service to provide Internet2 QoS", K.M. Nichols, Proceedings of the First Internet2 Joint Applications / Engineering QoS Workshop, 1998.

[nour02a] "Improving the performance of interactive TCP applications using service differentiation", W. Noureddine, F. Tobagi, Elsevier Computer Networks, vol. 40, no. 1, 2002.

[nour02b] "The Transmission Control Protocol. An introduction to TCP and a research survey", W. Noureddine, F. Tobagi, Technical Report, Stanford University, 2002.

[ns-2] UCB/LBL/VINT Network Simulator – ns (version 2), http://www.isi.edu/nsnam/ns.

[padh98a] "Modeling TCP throughput: a simple model and its empirical validation", J. Padhye, V. Firoiu, D. Towsley, J. Kurose, ACM SIGCOMM Computer Communication Review, vol. 28, no. 4, 1998.

[pare93a] "A generalized processor sharing approach to flow control in integrated services networks: the single-node case", A.K. Parekh, R.G. Gallager, IEEE/ACM Transactions on Networking, vol. 1, no. 3, 1993.

[park96a] "On the relationship between file sizes, transport protocols, and self-similar network traffic", K. Park, G.T. Kim, M.E. Crovella, Proceedings of the IEEE International Conference on Network Protocols (ICNP), 1996.

[paxs95a] "Wide area traffic: the failure of Poisson modeling", V. Paxson, S. Floyd, IEEE/ACM Transactions on Networking, vol. 3, no. 3, 1995.

[perr96a] "Call admission control schemes: a review", H.G. Perros, K.M. Elsayed, IEEE Communications Magazine, vol. 34, no. 11, 1996.

[prof01b] "Carrier Service-Level Agreements", The International Engineering Consortium, Web Proforum Tutorials IEC, http://www.iec.org, 2001.

[qiu01a] "Measurement-based admission control using aggregate traffic envelopes", J. Qiu, E. W. Knightly, IEEE/ACM Transactions on Networking, vol. 9, no. 2, 2001.

[reed00a] "The End of the End-to-End argument", D.P. Reed, http://www.reed.com/Papers/endofendtoend.html, 2000.

[rfc793] "Transmission Control Protocol", J. Postel, RFC 793, 1981.

[rfc1122] "Requirements for Internet hosts - communication layers", R. Braden, RFC 1122, 1989.

[rfc1633] "Integrated services in the Internet architecture: an overview", R. Braden, D. Clark, S. Shenker, RFC 1633, 1994.

[rfc1945] "Hypertext Transfer Protocol -- HTTP/1.0", T. Berners-Lee, R. Fielding, H. Frystyk, RFC 1945, 1996.

[rfc2018] "TCP selective acknowledgment options", M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, RFC 2018, 1996.

[rfc2205] "Resource ReSerVation protocol (RSVP) -- Version 1 functional specification", R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, RFC 2205, 1997.

[rfc2309] "Recommendations on queue management and congestion avoidance in the Internet", B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, RFC 2309, 1998.

[rfc2330] "Framework for IP Performance Metrics", V. Paxson, G. Almes, J. Mahdavi, M. Mathis, RFC 2330, 1998.

[rfc2386] "A framework for QoS-based Routing in the Internet", E. Crawley, R. Nair, B. Rajagopalan, H. Sandick, RFC 2386, 1998.

[rfc2474] "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", K. Nichols, S. Blake, F. Baker, D. Black, RFC 2474, 1998.

[rfc2475] "An architecture for Differentiated Services", S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, RFC 2475, 1998.

[rfc2581] “TCP Congestion Control”, M. Allman, V. Paxson, W. Stevens, RFC 2581, 1999.

[rfc2597] “Assured Forwarding PHB group”, J. Heinanen, F. Baker, W. Weiss, J. Wroclawsky, RFC 2597, 1999.

[rfc2598] “An Expedited Forwarding PHB”, V. Jacobson, K. Nichols, K. Poduri, RFC 2598 (obsoleted by RFC 3246), 1999.

[rfc2616] “Hypertext Transfer Protocol -- HTTP/1.1”, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, RFC 2616, 1999.

[rfc2638] “A two-bit Differentiated Services architecture for the Internet”, K. Nichols, V. Jacobson, L. Zhang, RFC 2638, 1999.

[rfc2702] “Requirements for Traffic Engineering Over MPLS”, D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus, RFC 2702, 1999.

[rfc2883] “An extension to the selective acknowledgement (SACK) option for TCP”, S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, RFC 2883, 2000.

[rfc2988] “Computing TCP's retransmission timer”, V. Paxson, M. Allman, RFC 2988, 2000.

[rfc3031] “Multiprotocol Label Switching architecture”, E. Rosen, A. Viswanathan, R. Callon, RFC 3031, 2001.

[rfc3086] “Definition of Differentiated Services Per Domain Behaviors and rules for their specification”, K. Nichols, B. Carpenter, RFC 3086, 2001.

[rfc3168] “The addition of Explicit Congestion Notification (ECN) to IP”, K. Ramakrishnan, S. Floyd, D. Black, RFC 3168, 2001.

[rfc3209] “RSVP-TE: Extensions to RSVP for LSP Tunnels”, D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, RFC 3209, 2001.

[rfc3246] “An Expedited Forwarding PHB (Per-Hop Behavior)”, B. Davie, A. Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, D. Stiliadis, RFC 3246, 2002.

[rfc3270] “Multi-Protocol Label Switching (MPLS) support of Differentiated Services”, F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen, RFC 3270, 2002.

[rfc3517] “A conservative selective acknowledgment (SACK) – based loss recovery algorithm for TCP”, E. Blanton, M. Allman, K. Fall, L. Wang, RFC 3517, 2003.

[rfc3550] “RTP A transport protocol for real time applications”, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RFC 3550, 2003.

[rfc3649] “HighSpeed TCP for Large Congestion Windows”, S. Floyd, RFC 3649, 2003.

[rfc3782] “The NewReno modification to TCP's Fast Recovery algorithm”, S. Floyd, T. Henderson, A. Gurtov, RFC 3782, 2004.

[rfc4594] “Configuration guidelines for DiffServ service classes”, J. Babiarz, K. Chan, F. Baker, RFC 4594, 2006.

[robe04b] “Internet traffic, QoS, and pricing”, J.W. Roberts, Proceedings of the IEEE, vol. 92, no. 9, 2004.

[rsvp] IETF Resource ReSerVation Protocol (RSVP) Working Group, http://www.ietf.org/html.charters/rsvp-charter.html

[salt84a] “End-to-end arguments in system design”, J.H. Saltzer, D.P. Reed, D.D. Clark, ACM Transactions on Computer Systems, vol. 2, no. 4, 1984.

[schm97a] "Quality of service - an overview", J. Schmitt, L. Wolf, Technical Report TR-KOM-1997-01, Darmstadt University of Technology, 1997.

[shen95a] "Fundamental design issues for the future Internet", S. Shenker, IEEE Journal on Selected Areas in Communications, vol. 13, no. 7, 1995.

[shio99a] "Overview of measurement-based connection admission control methods in ATM networks", K. Shiomoto, N. Yamanaka, T. Takahashi, IEEE Communications Surveys & Tutorials, vol. 2, no. 1, 1999.

[siva00c] "Achieving per-flow weighted rate fairness in a core stateless network", R. Sivakumar, T. Kim, N. Venkitaraman, V. Bharghavan, Proceedings of the IEEE Conference on Distributed Computing Systems, 2000.

[siva99b] "The Corelite QoS architecture: providing a flexible service model with a stateless core", R. Sivakumar, N. Venkitaraman, T. Kim, S. Lu, T. Nandagopal, V. Bharghavan, Research report, Illinois Mobile Environments Laboratory (TIMELY) research group at the University of Illinois at Urbana Champaign, 1999.

[stil96a] "Traffic scheduling in packet-switched networks: analysis, design, and implementation", D. Stiliadis, PhD thesis, University of California Santa Cruz, 1996.

[stoi03a] "Core-stateless fair queuing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks", I. Stoica, S. Shenker, H. Zhang, IEEE/ACM Transactions on Networking, vol. 11, no. 1, 2003.

[stoi99a] "Providing guaranteed services without per flow management", I. Stoica, H. Zhang, ACM SIGCOMM Computer Communication Review, vol. 29, no. 4, 1999.

[tcpwest] TCP Westwood homepage at the University of California, http://www.cs.ucla.edu /NRL/hpi/tcpw/.

[teit99a] "Internet 2 Qbone: building a testbed for Differentiated Services", B. Teitelbaum, S. Hares, L. Dunn, R. Neilson, V. Narayan, F. Reichmeyer, IEEE Network, vol. 13, no. 5, 1999.

[thom97a] "Wide-area Internet traffic patterns and characteristics", K. Thompson, G.J. Miller, R. Wilder, IEEE Network, vol. 11, no. 6, 1997.

[tian05a] "TCP in wireless environments: problems and solutions", Y. Tian, K. Xu, N. Ansari, IEEE Communications Magazine, vol. 43, no. 3, 2005.

[trim01a] "A management and control architecture for providing IP differentiated services in MPLS-based networks", P. Trimintzios, I. Andrikopoulos, G. Pavlou, P. Flegkas, D. Griffin; P. Georgatsos, D. Goderis, Y. T'Joens, L. Georgiadis, C. Jacquenet, R. Egan, IEEE Communications Magazine, vol. 39, no. 5, 2001.

[tura01a] "Load Control: congestion notifications for real-time traffic", Z.R. Turányi, L. Westberg, Proceedings of the IFIP Working Conference on Performance Modeling and Evaluation of ATM & IP Networks (ATM&IP), 2001.

[umla07a] "Experiences with the ns-2 Network Simulator – explicitly setting seeds considered harmful", M. Umlauft, P. Reichl, Proceedings of the Wireless Telecommunications Symposium (WTS), 2007.

[veci99a] "Adaptive video on demand service on RSVP capable network", C. Veciana-Nogués, J. Domingo-Pascual, Proceedings of the European Conference on Multimedia Applications, Services and Techniques, 1999.

[vila04a] "Dynamic management and restoration of virtual paths in broadband networks based on distributed software agents", P. Vilà, PhD thesis, University of Girona, 2004.

[voge95a] "Distributed multimedia and QoS: a survey", A. Vogel, B. Kerherve, G.V. Bochmann, J. Gecsei, IEEE Multimedia, vol. 2, no. 2, 1995.

[wang00a] "Resource allocation for elastic traffic: architecture and mechanisms", Z. Wang, A. Basu, Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), 2000.

[wang97a] "User-Share Differentiation (USD) Scalable bandwidth allocation for differentiated services", Z. Wang, Internet draft draft-wang-diff-serv-usd-00.txt, 1998.

[zhan91a] "Virtual Clock: a new traffic control algorithm for packet-switched networks", L. Zhang, ACM Transactions on Computing Systems, vol. 9, no. 2, 1991.

[zhan95a] "Service disciplines for guaranteed performance service in packet-switching networks", H. Zhang, Proceedings of the IEEE, vol. 83, no. 10, 1995.

[zhan98c] "A scalable resource management framework for Differentiated Services", L. Zhang, Proceedings of the First Internet2 Joint Applications / Engineering QoS Workshop, 1998.

[xiao00a] "Providing Quality of Service in the Internet", X. Xiao, PhD thesis, Department of Computer Science and Engineering, Michigan State University (USA), 2000.

[xiao99a] "Internet QoS: a big picture", X. Xiao, L. M. Ni, IEEE Network, vol. 13, no. 2, 1999.

[xu04a] "Binary increase congestion control for fast long-distance networks", L. Xu, K. Harfoush, I. Rhee, Proceedings of the IEEE Infocom, 2004.

[zhu06a] "Statistical connection admission control framework based on achievable capacity estimation", H. Zhu, V.O. K. Li, Z. Ma, M. Zhao, Proceedings of the IEEE International Conference on Communications (ICC), 2006.

Appendix A: Details of the simulation in ns-2

In this appendix we describe several details of the implementation of the schemes in ns-2, a discrete event simulator, which is open-source and object-oriented, with the core written in C++ and an OTcl interpreter as front-end [ns-2]. Firstly, we describe the initial installation of the simulator over which we have worked. Then we deal with the implementation of the 1st scheme, the modified TCP and the 2nd scheme. We briefly explain the simulator components that create each functional block in a simulation and the files related to its implementation (new files we have added or existing ones we have modified). Finally we also describe the different tasks carried out by the set of simulation scripts, shell scripts and C programs that we have created in order to generate the simulation and obtain the results.

Initial installation

We used the simulator version 2.1b8a (ns-2.1b8a) plus its updated daily version (or "snapshot") of February 22, 2002 (ns-2-snapshot-20020222). The ns-2 simulator requires several external components that were installed using the "allinone" package of 2.1b8a version. Besides ns-2.1b8a, this package contained Tcl/Tk (version 8.3.2), OTcl (version 1.0a7), TclCL (version 1.0b11), nam (version1.0a10), xgraph (version 12.1), and others. We used the C and C++ compilers in its version 2.96.

Both schemes have been implemented within the Diffserv module of the simulator, which allows traffic conditioning mechanisms (traffic metering and marking) and class-based queues to be created. The mark that identifies a packet class is called "code point", a type of traffic conditioning mechanism is called "policy" and a specific traffic conditioner of a given "policy" is called "policer" ("policy" and "policer" are linked to queues at ingress routers).

One of the first changes we made in the Diffserv module was to modify the file trace.cc (in ~ns/trace) so that the traces of the simulator show the "code point" (the mark) of each packet.

1st scheme

We implemented the different functional blocks of our 1st scheme (see also Subsection 5.3.2):

– The FIFO queue with priority discarding for the five packet classes (in all routers) is created by the new queue *dsCBDP* (CBDP stands for Class-Based Discarding Priority), in the two forms of *dsCBDP/edge* (for only ingress routers, which also perform traffic conditioning) and *dsCBDP/core* (for only core routers). It is implemented in the new files dsCBDP.cc, .h, dsCBDPq.cc, .h, dsCBDPedge.cc, .h, dsCBDPcore.cc, .h (in ~ns/diffserv) and in the existing file ns-default.tcl (in ~ns/tcl/lib). If the new arriving packet exceeds the queue's limit, the queue discipline searches for a packet with the highest discarding priority and discards it. The queue is configured in the same way as other existing queues of the Diffserv module: firstly, the number of physical queues (one in our case) and the number of virtual queues for each physical queue (one for each packet class) are defined; and secondly, each packet class, identified by a "code point" (or mark), is assigned to a virtual queue. Traffic conditioning mechanisms can then be assigned to queues *dsCBDP/edge* at ingress routers. The file ns-default.tcl contains the default values for *dsCBDP*.

– The per-flow throughput meter (at the egress routers) is created by the new queue *SFQmesuradora* when its variable *mode_t* is set to 0 (*mode_t* is used to indicate which of the two schemes is being used). It is implemented in the new files sfqMesuradora.cc, .h (in ~ns/queue) and in the existing file ns-default.tcl (in ~ns/tcl/lib). The queue counts the total received bytes of each flow starting from the arrival of the first packet during the chosen measurement duration, and then it calculates the corresponding flow's throughput. A flow is identified by the source and destination nodes and the source and destination ports. The file ns-default.tcl contains the default values for the *SFQmesuradora*.
– The sending rate meter, the marker and the admission controller (at the ingress routers) all together are created by two new types of "policy" (or types of traffic conditioning), *TbMbacBe* and *TSW2CMbacBe*. They are implemented in the existing files dsPolicy.cc, .h (in ~ns/diffserv). One of its parameters is an *SFQMesuradora* at the egress, from which they obtain (after the corresponding egress-ingress packet delay) the per-flow throughput measurement for the AC decision (and also the value of *mode_t* that indicates which scheme is being used). Both policies mark the flow's packets as RIN or ROUT during the AC phase, and after the AC decision as AIN or AOUT if the flow is accepted, or as BE if it is rejected. A flow is identified by the source and destination nodes and the source and destination ports. We use *TbMbacBe* for the modified TCP sources and *TSW2CMbacBe* for the "standard" TCP sources. For the decision of whether a packet is *in* or *out*, the policy *TbMbacBe* uses the token bucket algorithm and the policy *TSW2CMbacBe* uses the TSW algorithm and the probabilistic marker. These policies are linked to queues at ingress routers (in our case, *dsCBDP/edge*) and configured in the same way as other existing policies of the Diffserv module: firstly, the types of "policy" to be used are defined; and secondly, specific "policers" (of the previously defined "policy" types) for each flow are defined.

TCPs

We implemented the modified TCP source and the user's impatience in all TCP sources (see also Subsection 5.3.2):

– The modified TCP source is created by setting the new TCP variables *with_burst_cbr_*, *with_min_*, *with_noflowcontrol_* to 1 (if they are set to 0, which are the default values, the TCP source behaves as the "standard"). It is implemented in the existing files tcp.cc, .h, tcp-reno.cc, .h, tcp-newreno.cc, .h (in ~ns/tcp) and ns-default.tcl (in ~ns/tcl/lib). The TCP source is modified in order to avoid the short-term fluctuations of the sending rate and to send at a minimum rate. By setting *with_burst_cbr_* and *with_noflowcontrol_*, the TCP source does not send a burst of packets according to the congestion window but rather it sends one packet at a certain time Δt without stopping, i.e., it sends continuously. The variations of Δt are inversely proportional to the variations of the window w caused by "standard" TCP rate-adaptive algorithms (we use TCP NewReno). By setting *with_min_*, the congestion window is never smaller than a desired minimum value defined by the existing variable *windowInit_*, so that Δt has a maximum value, and therefore, the sending rate has a minimum value. Moreover, packet retransmission from the last acknowledged packet is triggered by the usual circumstances (i.e., when the corresponding ACK packet is not received during a timeout period or when three duplicated ACKs of a previous packet are received), and also when the packet sequence number reaches the end value and the corresponding ACK has not been received yet. The file ns-default.tcl contains the default values for the three new TCP variables.
– TCP sources can be asked to stop at a given time by the new Tcl command *abort*. This is implemented in the existing files tcp.cc, .h (in ~ns/tcp). Using this command we are able to consider the users' impatience, since we can stop the TCP source before the file transfer finishes if the transfer time is too high.

The "standard" TCP versions NewReno and SACK were already available in the initial installation of the simulator, while the implementation of TCP Westwood was obtained from the TCP Westwood homepage [tcpwest].

2nd scheme

We implemented the different functional blocks of our 2nd scheme in a similar way to our 1st scheme (see also Subsection 6.3.2):

- As in our 1st scheme, the FIFO queue with priority discarding for the four packet classes (in all routers) is created by the new queue *dsCBDP*.
- The per-aggregate throughput meter (at the egress routers) is created by the same new queue *SFQmesuradora* that we used in the 1st scheme, but its variable *mode_t* is set to 2. The queue periodically counts the total received bytes during the chosen measurement duration of classes AOUT, BE and PR from a given LP, and then it calculates the corresponding throughput.
- The sending rate meter, the marker and the admission controller (at the ingress routers) all together are created by the same new type of "policy" *TSW2CMbacBe* that we used in the 1st scheme for the "standard" TCP sources. From its parameter *SFQMesuradora*, it obtains (after the corresponding egress-ingress packet delay) the per-aggregate throughput measurement for the AC decision (and also the value of *mode_t* that indicates which scheme is being used). Once the AC decision is made, the flow's packets are marked as AIN or AOUT if the flow is accepted, or as BE if it is rejected.

Generating the simulation and obtaining results

Finally we created a set of ns-2 simulation scripts in OTcl, shell scripts and C programs for generating the simulation and obtaining the results, and more specifically, for carrying out the following tasks:

- Creating the network topology, queues, traffic conditioners, admission controllers, etc., for each simulation scenario.
- Random generation of the file size and the starting time of each TCP flow (for a given LP) according to a Poisson arrival process and the Pareto distribution respectively. After an initial simulation transient phase, a minimum of 10,000 flows per simulation was generated. We have not used the random generator provided by the ns-2 simulator in its version 2.1b8a, because it has been shown to have several weaknesses that were later solved in new versions (see a recent study in [umla07a] for more details). Instead we used an external C program with the combined multiple recursive random generator proposed by L'Ecuyer, with C code extracted from [law00a].
- Processing simulation data (the existing simulator traces and new added ones) to obtain per-flow information such as if the flow completes the file transfer or is aborted by the impatient user, the throughput obtained (the total received packets divided by the flow's lifetime), the AC decision, the number of sent packets, the number of duplicated packets and others. From this information the set of satisfied flows (generated after the initial simulation transient phase) are determined. In addition, we can also obtain the frequency distribution of throughput for a given set of flows (e.g., for the accepted flows), percentages of satisfied flows over the total accepted flows, etc.
- Processing simulation data corresponding to the set of satisfied flows to calculate the three performance parameters for each LP, i.e., the average "total" satisfied traffic load, the average "minimum" satisfied traffic load, and the average throughput of satisfied flows. The average throughput of satisfied flows is simply obtained by averaging the set of throughput

values of satisfied flows. The average total and minimum satisfied traffic loads require more elaborate processing: from the simulator traces, we filter the packets of the aggregation of satisfied flows, distinguishing between the *in* and *out* packets according to their mark, and from that, we calculate the two corresponding time averages. All these average values are obtained by averaging over the simulation time, but without considering the initial period of the simulation transient phase.

– Generating 10 independent replications of each simulation, where independence is achieved by using different seeds of the random generator, and calculating the sample mean and 95% confidence intervals of the three performance parameters to estimate their mean value [law00a].

Appendix B: Related author's publications

In this appendix we list the international journals, proceedings of international conferences and research reports where this work has been published.

International journals

"A guaranteed minimum throughput service for TCP flows using measurement-based admission control", Lluís Fàbrega, Teodor Jové, Pere Vilà, José Marzo, **International Journal of Communication Systems** (IJCS), vol. 20, no. 1, 2007 (ISI Impact Factor 0.225).

"Throughput guarantees for TCP flows in a network based on packet classes using edge-to-edge per-flow measurements", Lluís Fàbrega, Teodor Jové, Pere Vilà, José Marzo, **Simulation**, vol. 82, no. 6, 2006 (ISI Impact Factor 0.400).

"A packet class-based scheme for providing throughput guarantees to TCP flows", Lluís Fàbrega, Teodor Jové, Pere Vilà, José Marzo, Proceedings of the 5th IEEE International Workshop on IP Operations & Management (IPOM'05) in **Lecture Notes in Computer Science** (LNCS), vol. 3751, 2005 (ISI Impact Factor 0.402).

Proceedings of international conferences

"Admission control for TCP flows using packet classes and edge-to-edge measurements of aggregates", Lluís Fàbrega, Teodor Jové, Pere Vilà, José Marzo, Liliana Carrillo, Proceedings of the **IEEE International Conference on Communications** (ICC), 2006.

"A scheme for a guaranteed minimum throughput service based on packet classes", Lluís Fàbrega, Teodor Jové, Pere Vilà, José Marzo, Proceedings of the **International Symposium on Performance Evaluation of Computer and Telecommunication Systems** (SPECTS), 2005.

"Throughput guarantees for elastic flows using end-to-end admission control", Lluís Fàbrega, Teodor Jové, Yezid Donoso, Proceedings of the **IFIP/ACM Latin America Networking Conference** (LANC), 2003.

"An admission control approach for elastic flows in the Internet", Lluís Fàbrega, Teodor Jové, Antonio Bueno, José L. Marzo, Proceedings of the **IFIP Working Conference On Performance Modelling And Evaluation of ATM & IP Networks** (ATM&IP), 2001.

Research reports

"A proposal of an admission control method for the Assured Service in the Internet", Lluís Fàbrega, Teodor Jové, Research report IIiA 02-16-RR, **Institute of Informatics and Applications (IIiA) at the University of Girona (UdG)**, 2002.

"End-to-end admission control for a guaranteed minimum throughput service", Lluís Fàbrega, Teodor Jové, José Marzo, Pere Vilà, Research report IIiA 02-15-RR, **Institute of Informatics and Applications (IIiA) at the University of Girona (UdG)**, 2002.