UNIVERSITY OF MINNESOTA

This is to certify that I have examined this bound copy of a doctoral thesis by

Zhenhai Duan

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Zhi-Li Zhang and David Hung-Chang Du
Name of Faculty Advisers

_____
Signature of Faculty Advisers

_____
Date

GRADUATE SCHOOL

On Scalable Support of Quality of Services in the Internet

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Zhenhai Duan

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Zhi-Li Zhang and David Hung-Chang Du
Faculty Advisers

June 2003

This work was supported in part by the National Science Foundation under the grants ANI-0073819, ITR-0085824, and CAREER Award NCR-9734428. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the authors and do not necessarily reflect the views of the National Science Foundation.

To My Parents

Abstract

The Internet currently supports a best-effort connectivity service. There has been an increasing demand for the Internet to support a certain degree of quality of service (QoS) to meet various service requirements from different networking applications and to better utilize the networking resources. However, supporting QoS in the Internet has also raised great concerns about the scalability of any QoS solution; therefore, the QoS deployment on the Internet has been hindered.

This dissertation addresses the scalability issues in supporting QoS from two complementary aspects, namely the packet forwarding data plane and the network resource management control plane. On the packet forwarding data plane, a virtual time framework is proposed as a unifying packet scheduling framework to provide *scalable* support for guaranteed services in the Internet. In this framework, Internet core routers do not need to maintain any per-flow state and do not perform any per-flow operations. Consequently, they are able to handle a large number of simultaneous flows. The key notion in the virtual time framework is a *virtual timestamp*, which is initialized at network edge routers and referred and/or updated by network core routers, depending on the service granularity supported by the network. Several new *core stateless* packet scheduling algorithms are designed and analyzed to illustrate how both aggregate and per-flow service guarantees can be supported within this same framework. This is critical for the Internet to continue its evolution. Moreover, we investigate the cost-performance trade-offs in supporting QoS in the Internet by studying the QoS provisioning power of these packet scheduling algorithms.

On the network resource management control plane, two scalable bandwidth broker architectures are designed and investigated. The first one is a centralized bandwidth broker architecture, which is built upon the core stateless packet scheduling algorithms we designed. By conducting admission controls on a per-path basis instead of on a "hop-by-hop" basis, this bandwidth broker architecture significantly reduces the complexity of the admission control algorithm; therefore, it improves the scalability of existing bandwidth broker architectures. To further improve its scalability, a hierarchical bandwidth broker architecture is designed. In this architecture, multiple edge bandwidth brokers are deployed in a network, along with the conventional centralized bandwidth broker. Edge bandwidth brokers handle the flow admission control and resource management functionalities for certain pre-defined paths. They interact with the centralized bandwidth broker for allocating and de-allocating trunk bandwidth along the paths. In this way, the centralized bandwidth broker only needs to handle coarser time scale trunk bandwidth requests from edge bandwidth

brokers. Consequently, its scalability is greatly improved.

Finally, to provide real end-to-end QoS support and to facilitate the creation and deployment of *value-added services* such as VoIP, Video-on-Demand, and other emerging QoS-sensitive services over the Internet, an architecture called the *service overlay network* (SON) is proposed. Special servers, called service gateways, are deployed at certain strategically selected locations over the Internet to aid the data forwarding and resource management. The bandwidth provisioning problem for a service overlay network is mathematically formulated and investigated, taking into account various factors such as SLA, QoS, traffic demand distributions, and bandwidth costs. Analytical models and approximate solutions are developed for both static and dynamic bandwidth provisioning, which provide useful guidelines on how a SON should be provisioned to stay profitable.

Acknowledgments

I am very grateful to Professor Zhi-Li Zhang, who is my academic advisor. He has given me tremendous help during my graduate years at the University of Minnesota. I benefited greatly from the countless discussions with him about my research. Moreover, he is also a role model for how to be a critical thinker and an active researcher. I would like to thank him for his guidance, support, and encouragement.

I am also grateful to Professor David Hung-Chang Du, who is my academic co-advisor. His sagacious perception and critical insights in research have pushed me to think deeper and refine my dissertation work. I also want to thank Professor Andrew Odlyzko for taking his precious time to read through this dissertation and provide valuable feedback. I would like to thank Professor Yongdae Kim for serving on my final doctoral oral examination committee. I would also like to take this opportunity to thank Professor Michael Taaffe and Professor Ravi Janardan for serving on the committee for my preliminary doctoral oral examination.

Dr. Mooi Choo Chuah provided me with an internship opportunity at Bell-Labs during the Summer of 1998; Dr. Yiwei Thomas Hou provided me with internship opportunities at Fujitsu Labs of America during the Summer of 2000 and the Spring and Summer of 2001. I would like to thank them for providing me with these valuable opportunities that broadened my vision of research, and that will certainly continue to benefit me later in my research career.

The other members of the network group at the University of Minnesota have made my graduate life more joyous and easier. I treasure the joyful time we spent at the group get-together parties, lunches, and the other activities, and the "stressful" time we spent on discussing research projects. I want to especially thank James Beyer, Jaideep Chandrashekar, Baek-Young Choi, Changho Choi, Yingfei Dong, Dingshan He, Jeffrey Krasky, Ewa Kusmierek, Sanghwan Lee, Hyeran Lim, Yingping Lu, Wei-hsiu Ma (he was also my host when I first arrived in the USA), Srihari Nelakuditi, Kuai Xu, and Xu Zhang. James was the person that I ran to whenever I had some English grammar or general English writing problems. Over the years, he has also proofread many of my papers. I am deeply grateful to him.

Many friends at the University of Minnesota have helped me over the years. Without their generous help and care, my graduate life would not have been so colorful and easy. I want to take this opportunity to especially thank the following friends: Xiuzhen Cheng, Hua Dong,

Peiquan Li, Xiaojia Mary Li, Zining Li, Jian Liu, Kate Martin, Lu Ruan, Shuangyi Tang, Feng Wang, Hongyi Wang, and Xiaoyu Wu. Many other friends across the USA, Canada, and China have helped me and my parents, and encouraged me during my graduate years; I would especially like to thank Dongbo Bu, Chunmeng Cao, Lih-Yiu Chiou, Ji Fang, Ping Liu, Dewei Ma, Hong Wang, Rongxiang Wang, Wenqing Yuan, Jun Zhang, and Xiao Zhong.

I am deeply grateful to my parents, brother, and sisters for their unconditional love, long-standing support, and meticulous care. They have sacrificed greatly to support me in the pursuit of my dream. Without their love, support, and self-sacrifice, I would not have been able to reach this stage of my career.

Contents

List of Figures

List of Tables

Chapter 1

Overview

1.1 Introduction and Motivation

The Internet has been enjoying great success in the last ten years or so. In many aspects it grows at a rate of approximately 100 percent each year, for example, the number of hosts connected to the Internet and the volume of traffic carried by the Internet [15]. The unprecedented success of the Internet is largely due to its simple, "hour-glass" IP network protocol architecture (see Figure 1.1). In this architecture, the minimalist IP network protocol operates over a multitude of data link layer technologies and relegates sophisticated control to higher layer protocols operating at end systems [10].

Such a *thin-waist* IP network architecture enables Internet routers to be *stateless*. Routers only maintain certain highly aggregate routing information. They do not maintain any other forwarding state information. In particular, they do not maintain any per-flow state and do not perform any per-flow operations. The stateless property of the Internet architecture has some important implications for its performance. For example, the current Internet architecture is both *scalable and robust*. Because routers do not need to maintain any per-flow state and do not perform any per-flow operations, the operational complexity of Internet routers does not increase linearly with the number of flows present at the routers. Consequently, Internet routers are able to handle a large number of simultaneous flows. In this sense, we say the Internet architecture is scalable. Secondly, because routers do not maintain any per-flow state, after a link or router failure, packets of the flows being affected by the failure can be *automatically* routed around the failure point by an upstream router (if an alternative route exists). Therefore, end hosts are not even aware of such failures,

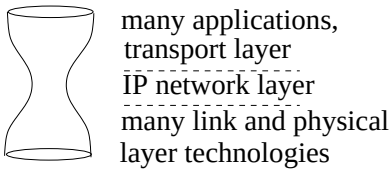


Figure 1.1: The IP hour-glass model.

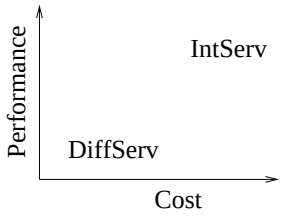


Figure 1.2: Cost-performance trade-offs in supporting QoS.

and no connection re-establishments are needed. In this sense, we say the current Internet architecture is robust.

On the other hand, the stateless Internet architecture also has some shortcomings. In particular, it cannot provide user traffic with any service guarantees in terms of delay or bandwidth. Note that, because routers do not maintain any per-flow state, they cannot differentiate packets from different flows and provide performance-demanding applications with more resources or better treatment. Indeed, the current Internet can only provide a *best-effort* connectivity service. It does not make any promise regarding the successful delivery of a packet, nor does it guarantee how long it takes for a packet to reach the destination. The best-effort service model is perhaps an adequate service model for the traditional Internet applications such as file transfers and E-mail. These applications are highly adaptive, they do not require stringent performance guarantees from the Internet.

However, as the Internet has transformed into a commercial information infrastructure, there has been a driving force to expand the current Internet service model to support more powerful services besides the base-line best-effort service [13, 14, 68, 77]. First of all, many real-time multimedia applications and mission-critical applications have emerged over the Internet, such as video streaming, Internet telephony, and online financial transactions, just to name a few. Different from the traditional Internet applications, these applications require a much more reliable and predictable Internet performance guarantee. Secondly, Internet service providers (ISPs) would like to have more control over how their network resources are used, for example, who can use how much bandwidth at what time. As we discussed above, all flows are treated in the same way in the current Internet architecture; flow differentiations are not supported. Moreover, to sustain a healthy long-term development, it is also very important for an ISP to explore new technologies to postpone and even reduce their network resource expansions, while still being able to provide a certain degree of performance guarantees.

To address these issues, the problem of providing Quality of Services (QoS) in the Internet has been the focus of both computer networking research and industrial communities for the last decade or so. Numerous QoS mechanisms have been carried out (see, e.g., [14, 22, 28, 29, 31, 32, 36, 37, 39, 41, 42, 43, 46, 47, 50, 57, 59, 60, 81, 83] and the references therein). Depending on the amount of control state maintained at Internet routers, we can classify QoS mechanisms into two categories: *stateful and stateless*. Stateful solutions need to maintain fine-grained *per-flow state at each Internet router* and perform certain per-flow operations. In contrast, stateless solutions preserve the stateless property of the current Internet architecture. Only coarse-grained aggregate state is maintained at Internet routers. The Integrated Service model (IntServ) [6] is a representative of a stateful QoS solution, while the Differentiated Service (DiffServ) model [4] is an example of a stateless QoS solution.

Stateful solutions such as IntServ provide very powerful and flexible QoS guarantees. In particular, per-flow rate and delay service guarantees can be supported in the IntServ framework. Compared with the stateful solutions, stateless solutions such as DiffServ can only provide *coarser-grained aggregate service guarantees*. Currently, it is not clear if such aggregate service guarantees are sufficient for supporting performance-demanding applications such as video streaming over the Internet. On the other hand, IntServ is a very costly solution. The operational complexity of routers in the IntServ framework increases linearly with the number of simultaneous flows at the routers. In contrast, DiffServ is much simpler and more scalable, given that no per-flow state is maintained by Internet core routers.

In a nutshell, stateful solutions such as IntServ can provide powerful and flexible QoS guarantees, but they are complex and costly. On the other hand, stateless solutions such as DiffServ are simpler and more scalable, but they can only provide limited aggregate performance guarantees. Figure 1.2 illustrates the broad trade-offs of these two approaches. More importantly, IntServ and DiffServ are all but two-point solutions. There is a plenty of design space that we have not explored, and many important questions in supporting QoS in the Internet remain unanswered. For example, what are the fundamental trade-offs in supporting QoS in the Internet in terms of performance and cost? Can we have a QoS solution that is as scalable as DiffServ, but is still able to provide performance guarantees as powerful and flexible as IntServ? Moreover, can we support both aggregate and per-flow service guarantees in the same framework? This is important for the Internet to continue its evolution.

1.2 Contributions

To address the above mentioned questions, and more importantly towards providing a complete *scalable QoS solution*, in this dissertation we propose and investigate a *virtual time service model* (VTSM). VTSM addresses the QoS provisioning problem from two complementary aspects, namely the packet forwarding data plane and the network resource management control plane. On the packet forwarding data plane, a virtual time framework is studied. The virtual time framework is core stateless. Internet core routers do not maintain any per-flow state and do not perform any per-flow operations. Moreover, both aggregate and per-flow service guarantees can be supported within this framework. On the resource management control plane, we design and investigate scalable bandwidth broker architectures to facilitate network resource management and admission control operations. To deliver real end-to-end QoS to end users, and to facilitate the deployment of new (QoS-sensitive) services over the Internet, a *service overlay network* architecture is also proposed and studied. Research results obtained in this dissertation work have been or will be partially reported in [24, 25, 26, 27, 85, 86, 87, 88, 89].

1.2.1 Scalable Packet Forwarding Date Plane

We propose and develop a virtual time framework (VTF) as a unifying packet scheduling framework to provide *scalable* support for guaranteed services. VTF is core stateless. Like DiffServ, Internet core routers do not need to maintain any per-flow state in this framework. However, different from DiffServ, both aggregate and per-flow service guarantees can be supported within this same framework. The key notion in VTF is the virtual timestamp, which is initialized at network edge routers and referred and/or updated by core routers, depending on the type of services supported by the network. We design and analyze several new *core stateless* packet scheduling mechanisms to illustrate how both aggregate and per-flow service guarantees can be supported within this framework. Moreover, we investigate the cost-performance trade-offs in supporting QoS in the Internet by studying the QoS provisioning power of these packet scheduling algorithms.

1.2.2 Scalable Network Resource Management Control Plane

To facilitate network resource management and flow admission control functions, two scalable bandwidth broker architectures are designed and investigated. The first one is a centralized bandwidth broker architecture, which is built upon the core stateless packet scheduling algorithms we designed. By conducting admission controls on a per-path basis instead of

on a "hop-by-hop" basis, this bandwidth broker architecture significantly reduces the complexity of the admission control algorithm; therefore, it improves the scalability of existing bandwidth broker architectures. To further improve its scalability, a hierarchical bandwidth broker architecture is designed. In this architecture, multiple edge bandwidth brokers are deployed in a network, along with the conventional centralized bandwidth broker. Edge bandwidth brokers handle the flow admission control and resource management functionalities for certain pre-defined paths. They interact with the centralized bandwidth broker for allocating and de-allocating trunk bandwidth along the paths. In this way, the centralized bandwidth broker only needs to handle coarser time scale trunk bandwidth requests from edge bandwidth brokers. Consequently, its scalability is greatly improved.

1.2.3 Service Overlay Networks

To provide real end-to-end QoS support, and to facilitate the creation and deployment of *value-added services* such as VoIP, Video-on-Demand, and other emerging QoS-sensitive services over the Internet, we propose an architecture called the *service overlay network* (SON). Special servers, called service gateways, are deployed at certain strategically selected locations over the Internet to aid the data forwarding and resource management. As a first step in designing a SON, we study the bandwidth provisioning problem for a service overlay network. We mathematically formulate the bandwidth provisioning problem, taking into account various factors such as SLA, service QoS, traffic demand distributions, and bandwidth costs. Analytical models and approximate solutions are developed for both static and dynamic bandwidth provisioning, which provide useful guidelines on how a SON should be provisioned to stay profitable.

1.3 Organization

The dissertation is structured into three parts. In this first part, we study the scalable packet forwarding data plane mechanisms. Specifically, in Chapter 3 we present aggregate packet forwarding schemes. Three different aggregate packet scheduling algorithms, namely FIFO, Static Earliest Time First (SETF), and Dynamic Earliest Time First (DETF), are studied. In Chapter 4, we investigate a virtual time reference system (VTRS) to illustrate how per-flow service guarantees can be supported in the Internet without maintaining per-flow state at network core routers. Several specific packet scheduling algorithms are also studied in this chapter to analyze the capability of VTRS to support per-flow service guarantees. In the second part of this dissertation, we present two scalable bandwidth bro-

ker architectures to facilitate network resource management and flow admission control operations. We will first study a centralized bandwidth broker architecture in Chapter 6 and then a hierarchical bandwidth-broker architecture in Chapter 7. In the third part of this dissertation, a service overlay network architecture is presented. We discuss its architecture and study the bandwidth provisioning problem for such a service overlay network in Chapter 8. In Chapter 9 we conclude this dissertation and discuss possible future research directions.

Part I

Scalable Packet Forwarding Data Plane

Chapter 2

Background and Overview

2.1 Background

The problem of Quality of Service (QoS) provisioning in packet-switched networks has been the focus of networking and telecommunication research communities for the last decade or so. Many new packet scheduling algorithms (see, e.g., [22, 59, 71, 83]) such as Virtual Clock (VC) and Weighted Fair Queueing (WFQ) have been proposed for the support of *QoS guarantees*. For example, it has been shown [30, 60] that in a network where WFQ schedulers (or VC schedulers) are employed at every router, end-to-end delay and bandwidth guarantees can be supported for each user traffic flow. Using these results as a reference model, the IETF has defined a *guaranteed service* [69] under its Integrated Services or IntServ architecture [6, 14], where end-to-end delay and bandwidth guarantees are provided for users on a *per-flow* (either individual or aggregate) basis. To support the IETF IntServ architecture, a signaling protocol, RSVP, for setting up end-to-end QoS reservation along a flow's path has also been proposed and standardized [7, 84].

Performing per-flow management inside the network, however, raises the important issue of *scalability*. Due to the complexities of per-flow operations both in the data plane and QoS control plane, the IntServ architecture *may not* scale well with the size of the Internet and the number of applications. As an alternative to per-flow based QoS provisioning, a different paradigm — the Differentiated Services or DiffServ — was later proposed and defined by the IETF [4, 5]. Under this paradigm, services are defined and implemented within individual administrative domains. To provide scalable QoS support, fine-grain user QoS control information is only maintained at the edge routers (i.e., ingress and egress routers)

of an administrative domain. The user traffic is appropriately *conditioned* (i.e., shaped) before injected into the network core. At core routers, no per-flow QoS state is maintained. User packets are processed based on a number of pre-specified *Per-Hop Behaviors* (PHBs) encoded by bit patterns carried inside a packet header that convey to core routers the desired levels of QoS support. (We will refer to these bit patterns, or the PHBs they embody, as the *packet state*.) End-to-end, *user-to-user* QoS support is provided through intra-domain QoS provisioning and inter-domain service agreement. These *control plane* functions can be performed, for example, by employing a *bandwidth broker* architecture [58]. The DiffServ paradigm greatly simplifies the data plane of the network core of a domain, thereby making it more scalable. On the other hand, the DiffServ architecture, as it is currently defined, aims to provide only *coarse-grain* QoS support to users. It remains to be seen whether such a service model would be sufficient to meet the potentially diverse user QoS requirements in the future.

Indeed, the exact QoS provisioning power that can be provided by DiffServ is still under great debate. For example, in the DiffServ framework [5], it is proposed that the simple FIFO packet scheduling be used to support the EF (*expedited forwarding*) per-hop behavior (PHB) [44]. Namely, at each router, EF packets from all users are queued at a single FIFO buffer and serviced in the order of their arrival times at the queue. In [44], it is claimed that EF can be "used to build a low loss, low latency, low jitter, assured bandwidth, end-to-end service." However, in a work by Charny and Le Boudec [11], it is shown that in order to provide guaranteed delay service using FIFO, the overall *network utilization level* must be limited to a small fraction of its link capacities. More specifically, in a network of FIFO schedulers, the *worst-case* delay at each router is bounded only when the network utilization level is limited to a factor smaller than $1/(H^* - 1)$, where H^*, referred to as the *network diameter*, is the number of hops in the longest path of the network. Furthermore, given the network utilization level $\alpha < 1/(H^* - 1)$, the worst-case delay bound is *inversely proportional* to $1 - \alpha(H^* - 1)$. Hence as the network utilization level α gets closer to the utilization bound $1/(H^* - 1)$, the worst-case delay bound approaches rapidly to infinity.

The elegant result of Charny and Le Boudec raises several interesting and important questions regarding the *design* and *provisioning power* of aggregate packet scheduling, or more generally, *the fundamental cost-performance trade-offs in supporting QoS in the Internet.*

For ease of explosition, in the following we will refer to the schemes that require per-flow state and perform per-flow operations at each Internet routers as *stateful* QoS solutions; whereas schemes that do not perform per-flow operations such as DiffServ as *stateless* QoS

solutions. From the above discussions, we see that stateful QoS solutions such as IntServ can provide powerful and flexible per-flow service guarantees, but they are complex and costly. On the other hand, stateless QoS solutions such as DiffServ are much more scalable, but they can only provide coarse-grained aggregate QoS support.

In an important piece of work, Stoica and Zhang, using the DiffServ paradigm and the novel notion of *dynamic packet state*, developed several techniques to support end-to-end *per-flow* bandwidth and delay guarantees *without per-flow QoS management*[73]. In the data plane, they designed a new (non-work-conserving) scheduling algorithm, called *Core Jitter Virtual Clock* or CJVC, to provide end-to-end per-flow delay guarantees *without per-flow scheduling states at core routers*. (Such scheduling algorithms are referred to as *core stateless*, in contrast to the conventional *stateful* scheduling algorithms such as VC or WFQ, where certain scheduling states must be maintained for each flow.) In the control plane, an *aggregate reservation estimation* algorithm is designed which eliminates the need of maintaining *per-flow QoS reservation states*. Instead, an *aggregate* QoS reservation state is maintained at each core router. A hop-by-hop signaling protocol, however, is still needed to set up QoS reservation for each flow along its path within a domain. The work by Stoica and Zhang is the first to demonstrate how per-flow service gurantees can be supported without requiring Internet core routers to maintain per-flow state and perform per-flow operations. However, in their work, only one single non-work-conserving packet scheduling is studied. It is not clear if the basic technique can be used in a more general, working-conserving packet scheduling environment. More importantly, the question regarding the fundamental cost-performance trade-offs in supporting QoS in the Internet still remain unanswered.

2.2 Virtual Time Framework

In this part of the dissertation, we propose and develop a virtual time framework (VTF) as a unifying packet scheduling framework to provide scalable support for guaranteed services. Similar to the DiffServ architecture, in VTF, we distinguish network edge routers from core routers. All per-flow state is maintained at network edge routers, core routers do not maintain any per-flow state and do not perform any per-flow operation. That is, the virtual time framework is core stateless. As illustrated in Figure 2.1, VTF has three key components: packet state, edge router mechanism, and core router mechanism. In the following we briefly discuss each of them one by one.

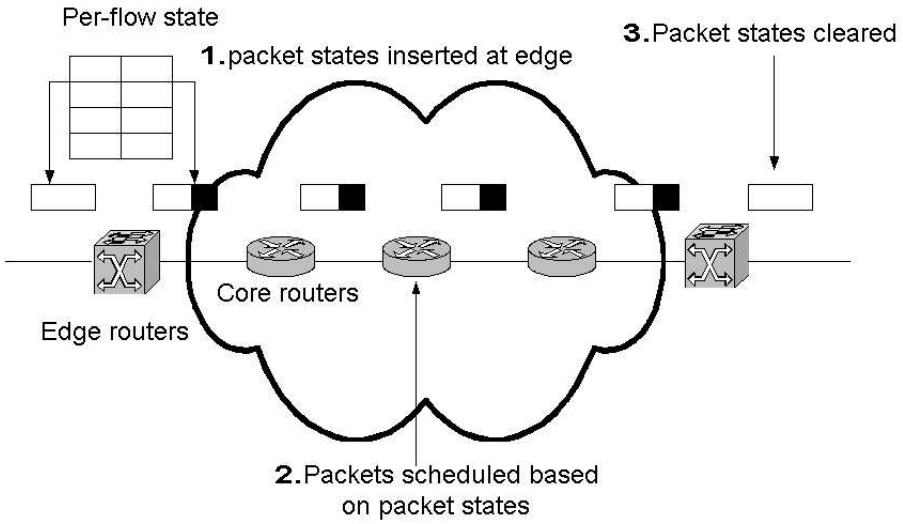


Figure 2.1: Illustration of Virtual Time Framework.

2.2.1 Packet State

When a packet is released into a network at an edge router, certain control state is inserted into the packet header by the edge router. The key packet state is a *virtual timestamp*, which is initialized as the packet releasing time at the network edge router. Depending on the type of services supported by the network, some additional control state may be carried in the packet header, such as the reservation rate of the flow that the packet belongs to. As we will see shortly, core routers rely on the virtual timestamps to schedule packets. Moreover, virtual timestamps may be updated within a network, again, depending on the service supported by the network.

2.2.2 Edge Router Mechanism

Edge routers maintain per-flow state. As we discussed above, they also insert certain packet state into the header of a packet when the packet is released into the network. Another important role of an edge router is to shape the traffic of a flow released into the network, to ensure that the amount of traffic of the flow released into the network satisfies certain condition.

2.2.3 Core Router Mechanism

Core routers rely on the virtual timestamp (and other control state) carried in packet headers to schedule packets. Depending the specific packet scheduling algorithm implemented in

the network (and therefore the service supported by the network), core routers may update the virtual timestamps carried in the packets.

In the next two chapters, we will present several specific implementations within this framework to illustrate its components and properties in detail. More specifically, in Chapter 3, we study two *new* classes of aggregate packet scheduling algorithms: the *static earliest time first* (SETF) and *dynamic earliest time first* (DETF) algorithms. In the class of SETF packet scheduling algorithms, packets are stamped with its *entry time at the network edge*, and they are scheduled in the order of their time stamps (i.e., their network entry times) inside the network core; the class of DETF packet scheduling algorithms work in a similar fashion, albeit with an important difference—the packet time stamps are *updated* at certain routers (hence the term *dynamic*). In Chapter 4, we design a novel *virtual time reference system* as a general architecture to support *per-flow* service guarantees. By studying these packet scheduling algorithms, we illustrate the fundamental trade-offs in supporting QoS in the Internet. Moreover, we demonstrate that both aggregate and per-flow service guarantees can be supported in the same framework.

Chapter 3

Supporting Aggregate Guaranteed Delay Services

3.1 Introduction

In this chapter, we illustrate how aggregate service guarantees can be supported in the virtual time framework. More importantly, we attempt to address the fundamental trade-offs in the design of aggregate packet scheduling algorithms and their provisioning power in support of (worst-case) *guaranteed delay* service. In particular, we study the relationships between the *worst-case edge-to-edge delay* (i.e., the maximum delay experienced by any packet across a network domain), the *maximum allowable network utilization level* and the *"sophistication/complexity"* of aggregate packet scheduling employed by a network.

Besides the conventional "TOS" or "DS" bits, we assume that *additional control information* may be carried in the packet header for scheduling purpose. By encoding certain *timing* information in the packet header, we design two *new* classes of aggregate packet scheduling algorithms: the *static earliest time first* (SETF) and *dynamic earliest time first* (DETF) algorithms. In the class of SETF packet scheduling algorithms, packets are stamped with its *entry time at the network edge*, and they are scheduled in the order of their time stamps (i.e., their network entry times) inside the network core; the class of DETF packet scheduling algorithms work in a similar fashion, albeit with an important difference—the packet time stamps are *updated* at certain routers (hence the term *dynamic*). In both classes, the *granularity* of timing information encoded in the packet state—as is determined by the *number of bits* used for packet state encoding—is a critical factor that affects the provisioning power

of aggregate packet scheduling.

The objective of our study is to use these two new classes (SETF and DETF) of aggregate packet scheduling algorithms, in addition to the simple FIFO discipline, to illustrate the fundamental trade-offs in aggregate packet scheduling: 1) *how with additional control information encoded in the packet state, and with added "sophistication/complexity" in aggregate packet scheduling, the worst-case edge-to-edge delay bound and the maximum allowable network utilization bound can be improved*; and 2) *how these performance bounds are affected by the number of bits available for packet state encoding*. Through analysis and numerical examples, we show that when packet time stamps are encoded with the *finest* time granularity, both the SETF and DETF packet scheduling algorithms can attain an arbitrary network utilization level (i.e., α can be arbitrarily close to 1). In other words, the maximum allowable network utilization bound is *independent of* the network diameter H^*. This is in contrast to the case of FIFO, where the maximum utilization level is bounded by $1/(H^* - 1)$. Furthermore, using the more complex DETF, the worst-case edge-to-edge delay bound is *linear in* H^*, whereas using the simpler SETF, the worst-case edge-to-edge delay bound is *inversely proportional to* $(1-\alpha)^{H^*}$. When packet time stamps are encoded using coarser granularity (i.e., the number of bits for packet state encoding is limited), the network utilization level is constrained by the time granularity. In addition, the worst-case edge-to-edge delay bound is increased. With the same number of bits, the more complex DETF packet scheduling algorithms have far superior performance over the simpler SETF algorithms.

The remainder of this chapter is organized as follows. In Section 3.2 we present the basic model and assumptions for our analysis. In Section 3.3, we re-establish the result in [11] using our approach. The two new classes of aggregate packet scheduling, SETF and DETF, are analyzed and the trade-offs discussed in Section 3.4 and Section 3.5, respectively. We summerize this chapter in Section 3.6.

3.2 Network Model and Assumptions

Consider a single network domain, as shown in Figure 3.1, where all traffic entering the network is shaped at the edge traffic conditioner before releasing into the network. No traffic shaping or re-shaping is performed inside the network core. We assume that all routers employ the same aggregate packet scheduling algorithm (e.g., FIFO) that performs packet scheduling using only certain bits (the *packet state*) carried in the packet header.

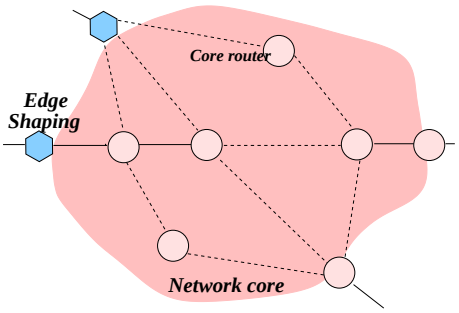


Figure 3.1: The network model.

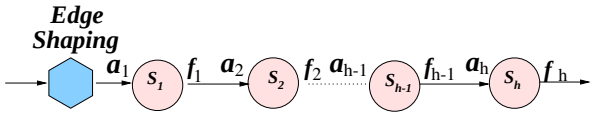


Figure 3.2: Packet's arrival time at and departure time from each scheduler.

No other *scheduling* information is used or stored at core routers. We refer to the scheduling mechanism employed at an outgoing link of a router[1] as a *scheduler*. Let C be the capacity of the corresponding outgoing link of a scheduler S. We will also refer to C as the capacity of the scheduler S. We denote the MTU (maximum transmission unit) of the link by L^{max}, then L^{max}/C is the transmission time of an MTU-sized packet. Define $\Delta = \max_{allS's}\{L^{max}/C\}$, i.e., Δ is the maximum transmission time of any packet in the network. We also assume that the path of any user flow is pre-determined, and fixed throughout its duration. Let H^* be the maximum number of hops in the paths that any user flow may traverse in the network. We refer to H^* as the *network diameter*.

Consider an arbitrary flow j traversing the network. The traffic of the flow is shaped at the network edge in such a manner that it conforms to a *token bucket regulated arrival curve* (σ^j, ρ^j) [20]: Let $A^j(t, t+\tau)$ denote the amount of the flow j traffic released into the network during a time interval $[t, t+\tau]$, where $t \geq 0$, $\tau \geq 0$; then $A^j(t, t+\tau) \leq \sigma^j + \rho^j\tau$. We control the overall network utilization level *by imposing a utilization factor* α on each link as follows. Consider an arbitrary scheduler S with capacity C. Let $\mathcal{F}$ denote the set of user flows traversing S. Then the following condition holds:

$$\sum_{j\in\mathcal{F}} \rho^j \leq \alpha C. \tag{3.1}$$

Clearly, $0 < \alpha \leq 1$. We will also refer to the utilization factor α as the *network utilization level* of a network domain. In addition to the link utilization factor α, we will also impose an overall bound $\beta \geq 0$ (in units of time) on the "burstiness" of flows traversing any scheduler S: $\sum_{j\in\mathcal{F}} \sigma^j \leq \beta C$. As we will see later, this *burstiness factor* β plays a less critical role in our analysis than the network utilization level α.

[1]For simplicity, we assume that output-queueing is used.

From the above edge shaping and network utilization constraints, we can obtain an important bound on the amount of traffic going through a given scheduler *that is injected at the network edge during any time interval*. Consider an arbitrary scheduler S with capacity C. For any time interval $[\tau, t]$, let $\mathring{A}_S(\tau, t)$ denote the amount of traffic injected into the network during the time interval $[\tau, t]$ that will traverse S (at perhaps some later time). Here we use $\mathring{A}$ to emphasize that $\mathring{A}_S(\tau, t)$ is *not* the traffic traversing S during the time interval $[\tau, t]$, *but* injected into the network at the network edge during $[\tau, t]$. Using the facts that $A^j(t, t+\tau) \leq \sigma^j + \rho^j \tau$ for all flows, $\sum_{j \in \mathcal{F}} \rho^j \leq \alpha C$ and $\sum_{j \in \mathcal{F}} \sigma^j \leq \beta C$, it is easy to show that

$$\mathring{A}_S(\tau, t) \leq \alpha C(t - \tau) + \beta C. \tag{3.2}$$

We refer to this bound as the *edge traffic provisioning condition* for scheduler S. As we will see later, the edge traffic provisioning condition is critical to our analysis of aggregate packet scheduling algorithms.

Now consider a packet p (of any flow) that traverses a path with $h \leq H^*$ hops. For $i = 1, 2, \ldots, h$, denote the scheduler at the ith hop on the path of packet p as S_i (see Figure 3.2). Let a_i^p and f_i^p represent, respectively, the time that packet p arrives at and departs[2] from scheduler S_i. For ease of exposition, throughout this chapter we assume that the propagation delay from one scheduler to another scheduler is zero. Hence $a_{i+1}^p = f_i^p$.

Note that a_1^p is the time packet p is released into the network (after going through the edge traffic conditioner), and f_h^p is the time packet p leaves the network. Hence $f_h^p - a_1^p$ is the cumulative delay that packet p experiences along its path, and is referred to as the *edge-to-edge* delay experienced by packet p. (Note that the delay experienced by a packet at the edge traffic conditioner is excluded from the edge-to-edge delay.) Define D^* to be the worst-case edge-to-edge delay experienced by any packet in the network, i.e.,

$$D^* = \max_{\text{all } p\text{'s}} \{f_h^p - a_1^p\}, \tag{3.3}$$

where in the above definition h is the number of hops on the path of packet p.

The key questions that we will address in the remainder of the chapter are: 1) given an aggregate packet scheduling algorithm, under what network utilization level α does an

[2]Throughout the chapter we adopt the following convention: a packet is considered to have arrived at a scheduler *only* when its last bit has been received, and it to have departed from the scheduler *only* when its last bit has been serviced.

upper bound on D^* exist? 2) how does this bound depend on the network utilization level α and the network diameter H^*? and 3) how these relationships are affected by the number of bits available for packet state encoding as well as the added "sophistication/complexity" in aggregate packet scheduling?

3.3 Network of FIFO Schedulers

In this section we re-establish the result of Charny and Le Boudec [11] for a network of FIFO schedulers *using a different approach*. Unlike [11] which uses an argument based on the worst-case per-hop delay analysis, in our approach we attempt to obtain a recursive relation for a_i^p's (or equivalently, f_i^p's) for any packet p. From this recursive relation we then derive an upper bound on the worst-case edge-to-edge delay D^*. As we will see later, this argument is quite general and powerful, and forms the basis of all the analyses in this chapter.

A key step in our analysis is to obtain an upper bound on the amount of traffic that is serviced by a scheduler between the arrival and departure of any packet p at the scheduler. This bound will allow us to establish a recursive relation between a_{i+1}^p and a_i^p. For this purpose, we introduce an important notation, τ^*, which is *the maximum time it takes for any packet to reach its last hop*. Formally,

$$\tau^* = \max_{\text{all } p\text{'s}} \{a_h^p - a_1^p\}. \tag{3.4}$$

Now consider a FIFO scheduler S of capacity C. Let a_S^p denote the time a packet p arrives at S, and f_S^p the time packet p departs from S. Define $Q(a_S^p)$ to be *the amount of traffic serviced by the scheduler S between $[a_S^p, f_S^p]$*. Note that since S is a FIFO scheduler, $Q(a_S^p)$ is exactly the amount of traffic queued at S at the arrival time of packet p (with packet p itself included). We have the following bound on $Q(a_S^p)$:

Lemma 1 *For a FIFO scheduler S of capacity C, we have*

$$Q(a_S^p) \leq \alpha C \tau^* + \beta C. \tag{3.5}$$

Proof: Let p^* be the *last* packet before packet p (itself inclusive) that when packet p^* arrives at scheduler S any packet p' in the queue (including the one in service) satisfies the

following condition:

$$a_1^{p'} \geq a_1^{p*}. \tag{3.6}$$

In other words, when packet p^* arrives at scheduler S, it is the "oldest" packet in the queue: namely, *all other packets currently in the queue entered the network no early than packet* p^*. We note that such a packet always exists—if no other packets satisfy (3.6), the packet that starts the current busy period certainly does. Let $a_S^{p^*}$ denote the time packet p^* arrived at scheduler S. By the definition of p^*, any packet that was either queued at scheduler S at time $a_S^{p^*}$ or arrived at scheduler S between $a_S^{p^*}$ and a_S^p must have entered the network during the time interval $[a_1^{p^*}, a_S^p]$. From (3.2), the amount of traffic carried by these packets is bounded above by $\alpha C(a_S^p - a_1^{p^*}) + \beta C$. Furthermore, since scheduler S is always busy during $[a_S^{p^*}, a_S^p]$, we have

$$\begin{aligned} Q(a_S^p) &\leq \alpha C(a_S^p - a_1^{p^*}) + \beta C - (a_S^p - a_S^{p^*})C \\ &= \alpha C(a_S^p - a_S^{p^*} + a_S^{p^*} - a_1^{p^*}) + \beta C - (a_S^p - a_S^{p^*})C \\ &= \alpha C(a_S^{p^*} - a_1^{p^*}) + \beta C + (\alpha - 1)(a_S^p - a_S^{p^*})C \\ &\leq \alpha C\tau^* + \beta C, \end{aligned}$$

the last step is because $a_S^{p^*} - a_1^{p^*} \leq \tau^*$ (3.4) and $\alpha \leq 1$. ∎

There is an intuitive explanation of the result in Lemma 1. Note that a FIFO scheduler services packets in the order of their arrival times at the scheduler, regardless of when they are released into the network. In particular, packets entering the network later than packet p can potentially be serviced earlier than packet p. Intuitively, packets that are queued at the time packet p arrives at scheduler S must have entered the network between $[a_S^p - \tau^*, a_S^p]$ and arrived at scheduler S before packet p. By the edge traffic provisioning condition (3.2), the amount of traffic carried by these packets is bounded by $\alpha C\tau^* + \beta C$. This intuitive argument is made rigorous in the proof of Lemma 1.

We now use Lemma 1 to derive a recursive relation for a_i^p's. Consider a packet p which traverses a path with h hops. The capacity of the ith scheduler on the path is denoted by C_i. Then by the definition of $Q(a_i^p)$, we have

$$a_{i+1}^p = f_i^p = a_i^p + Q(a_i^p)/C_i \leq a_i^p + \alpha\tau^* + \beta. \tag{3.7}$$

Recursively applying (3.7) and using the relation $f_i^p = a_{i+1}^p$, we have the following lemma.

Lemma 2 *Consider a packet p which traverses a path with h hops. Then, for $i = 1, 2, \ldots, h$, we have,*

$$f_i^p - a_1^p \leq i(\alpha\tau^* + \beta). \tag{3.8}$$

Using Lemma 2, we can establish the following main results for a network of FIFO schedulers.

Theorem 1 *Given a network of FIFO schedulers with a network diameter H^*, if the network utilization level α satisfies the condition $\alpha < \frac{1}{H^*-1}$, then $\tau^* \leq \frac{(H^*-1)\beta}{1-(H^*-1)\alpha}$. Furthermore, the worst-case edge-to-edge delay D^* is bounded above by*

$$D^* \leq \frac{H^*\beta}{1-(H^*-1)\alpha}. \tag{3.9}$$

Proof: The proof includes two parts. We first prove that for the given value of τ^*, (3.4) holds, i.e., no packets will experience a delay larger than τ^* before reaching the last hop. In the second part, we prove that the maximum edge-to-edge delay D^* is indeed bounded.

Part 1. Let $\tau^* = \frac{(H^*-1)\beta}{1-(H^*-1)\alpha}$. We claim that for any packet p which traverses a path with h hops, the following relationship holds for any $i \leq h$,

$$a_i^p - a_1^p \leq \tau^*. \tag{3.10}$$

Otherwise, assume that *for the first time in the system (the network domain)* a packet p^* violates the relationship at server $\mathcal{S}_{i^*}$ where $i^* \leq h$ (without loss of generality, we assumed here that the path that packet p^* traverses has h hops), i.e., $a_{i^*}^{p^*} - a_1^{p^*} > \tau^*$. Therefore (3.10) holds for all packets before the time $a_{i^*}^{p^*}$, in particular, we have,

$$a_{i^*-1}^{p^*} - a_1^{p^*} \leq \tau^*.$$

From Lemma 2, we have,

$$a_{i^*-1}^{p^*} - a_1^{p^*} \leq (i^*-1)(\alpha\tau^* + \beta).$$

Based on the result in Lemma 1 and the above equation, we get,

$$f_{i^*-1}^{p^*} \leq a_{i^*-1}^{p^*} + \alpha\tau^* + \beta \leq a_1^{p^*} + i^*(\alpha\tau^* + \beta). \tag{3.11}$$

After some algebra, we have,

$$f_{i^*-1}^{p^*} - a_1^{p^*} \leq \frac{(h-1)\beta}{1-(H^*-1)\alpha} \leq \tau^*.$$

Notice that $f_{i^*-1}^{p^*} = a_{i^*}^{p^*}$, we arrive at a contradition.

Part 2. In this part we prove that the worst-case edge-to-edge delay is bounded above by $D^* \leq \frac{H^*\beta}{1-(H^*-1)\alpha}$.

Consider an arbitrary packet p which traverses a path with h hops. From Lemma 2, we have,

$$f_h^p - a_1^p \leq h(\alpha\tau^* + \beta) \leq \frac{h\beta}{1-(H^*-1)\alpha} \leq \frac{H^*\beta}{1-(H^*-1)\alpha}.$$

■

Theorem 1 illustrates the provisioning power of a network of FIFO schedulers for support of guaranteed delay service: in order to provide a *provable* worst-case edge-to-edge delay bound, the maximum network utilization level must be limited below $1/(H^*-1)$. (We will refer to this bound as the *maximum allowable network utilization bound*). For example, with $H^* = 3$ (a "small" network), the maximum network utilization must be kept below 50% of all link capacities; with $H^* = 11$ (a relatively "large" network), the maximum network utilization must be kept below 10% of all link capacities. Furthermore, as the network utilization level gets closer to $1/(H^*-1)$, the worst-case edge-to-edge delay bound approaches infinity.

3.4 Network of Static Earliest Time First Schedulers

In this section we will design and analyze a new class of aggregate packet scheduling algorithms—the class of *static earliest time first* (SETF) algorithms. Using this class of aggregate packet scheduling algorithms, we will demonstrate how by adding some "sophistication/complexity" in aggregate packet scheduling—in particular, by encoding additional

control information in the packet header, we can improve the maximum allowable utilization bound, and reduce the provable worst-case edge-to-edge delay bound. Furthermore, we will discuss the performance trade-offs of SETF packet algorithms when a limited number of bits is used for packet state encoding.

The additional control information used by the class of SETF schedulers is a *(static) time stamp* carried in the packet header of a packet that *records the time the packet is released into the network* (after going through the edge traffic conditioner) at the network edge. Here we assume that all edge devices that time-stamp the packets use a *global* clock (in other words, the clocks at the edge devices are synchronized). We denote the time stamp of a packet p by ω_0^p. An SETF scheduler inside the network core schedules packets in the order of their time stamps, ω_0^p. Note that in the case of SETF, *the time stamp of a packet is never modified by any SETF scheduler, thus the term* static.

Depending on the time granularity used to represent the packet time stamps, we can design a class of SETF schedulers with different performance/complexity trade-offs. We use SETF(Γ) to denote the SETF packet scheduling algorithm where packet time stamps are represented with time granularity Γ. In particular, SETF(0) denotes the SETF packet scheduling algorithm where packet time stamps are represented with the *finest* time granularity, namely, packets are time-stamped with the *precise* time they are released into the network. Formally, for any packet p, we have $\omega_0^p = a_1^p$. For a more general SETF(Γ) scheduling algorithm where $\Gamma > 0$, we divide the time into slots of Γ time units each (see Figure 3.3): $t_n = [(n-1)\Gamma, n\Gamma), n = 1, 2, \ldots$. Packets released into the network are time-stamped with the corresponding time slot number n. In other words, packets that are released into the network within the same time slot (say, the time slot $t_n = [(n-1)\Gamma, n\Gamma)$) carry the same time stamp value, i.e., $\omega_0^p = n$. Therefore, packets released into the network during the same time slot at the network edge are *indistinguishable* by an SETF(Γ) scheduler inside the network core, and are serviced by the scheduler in a FIFO manner. We will show later that using coarser time granularity (i.e., larger Γ) can potentially reduce the number of bits needed to encode the packet time stamps, but at the expenses of degrading the performance bounds. In the following we will analyze SETF(0) first, since its analysis is easier to present and follow. The general SETF(Γ) will be studied afterwards in Section 3.4.2.

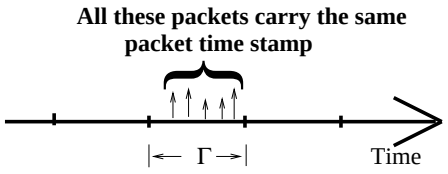


Figure 3.3: Time slots and packet time stamps.

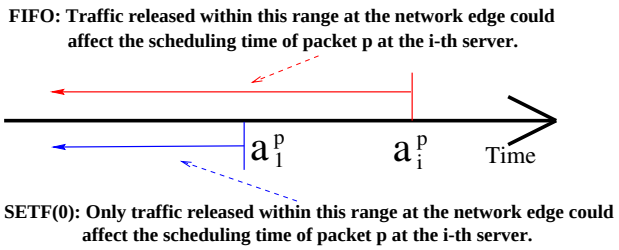


Figure 3.4: Illustration of the different behaviors of FIFO and SETF(0).

3.4.1 SETF with Finest Time Granularity: SETF(0)

In this section we first establish performance bounds[3] for SETF(0) and then discuss the packet state encoding issue.

3.4.1.1 Network Utilization and Edge-to-Edge Delay Bounds

We follow the same approach to establish performance bounds for a network of SETF(0) schedulers, as is employed for a network of FIFO schedulers in Section 3.3.

Consider an arbitrary SETF(0) scheduler S of capacity C. As in Section 3.3, let a_S^p and f_S^p denote, respectively, the time packet p arrives at and departs from S, and $Q(a_S^p)$ denote the amount of traffic serviced by the scheduler S between $[a_S^p, f_S^p]$. Note that unlike a FIFO scheduler, $Q(a_S^p)$ may be *not* equal to the amount of traffic queued at S at the arrival time of packet p. This is because a packet p' in the queue of scheduler S at the time of packet p arrival may have a time stamp $\omega_0^{p'} > \omega_0^p$. In addition, a packet p' arriving at S later than packet p (but before f_S^p) may have a time stamp $\omega_0^{p'} < \omega_0^p$, thus beginning service before packet p. Nonetheless, we can apply a similar argument as used in Lemma 1 to establish the following bound on $Q(a_S^p)$. To focus on the main message, the proof of the following lemma, and the proofs of all other lemmas and theorems thereafter, are delegated into the appendixes.

Lemma 3 *For an SETF(0) scheduler S of capacity C, we have*

$$Q(a_S^p) \leq \alpha C\{\tau^* - (a_S^p - a_1^p)\} + \beta C + L^{max}. \tag{3.12}$$

[3] A simplified version of SETF(0), under the name of *Longest-in-System* (LIS), is studied under a different but simpler network model in [2]. Our proof, inspired by [2], is more general than that in [2]. We also establish tighter performance bounds than [2].

Comparing Lemma 3 with Lemma 1, we see that the upper bound on $Q(a_S^p)$ for an SETF(0) scheduler is reduced by $\alpha C(a_S^p - a_1^p)$ amount from that for an FIFO scheduler. This is not surprising, since any packet that is released into the network after $a_1^p = \omega_0^p$ (the time packet p enters the network) will not take any service away from packet p at an SETF(0) scheduler (see Figure 3.4).

Now consider a packet p traversing a path with h hops, where the capacity of the ith scheduler on the path is C_i. From Lemma 3, the following recursive relations follow easily: for $i = 1, \ldots, h$,

$$a_{i+1}^p = f_i^p = a_i^p + \frac{Q(a_i^p)}{C_i} \leq (1-\alpha)a_i^p + \alpha(\tau^* + a_1^p) + \beta + \Delta.$$

Solving the above recursive equations, we have

Lemma 4 *Consider a packet p which traverses a path with h hops. Then for $i = 1, 2, \ldots, h$, we have,*

$$f_i^p - a_1^p \leq \tau^*\{1 - (1-\alpha)^i\} + \alpha^{-1}(\beta + \Delta)\{1 - (1-\alpha)^i\}. \tag{3.13}$$

Using Lemma 4, we can establish the following main results for a network of SETF(0) schedulers.

Theorem 2 *Consider a network of SETF(0) schedulers with a network diameter H^*. For $0 < \alpha < 1$, we have $\tau^* \leq \frac{\alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{H^*-1}\}}{(1-\alpha)^{H^*-1}}$. Moreover, the worst-case edge-to-edge delay D^* is bounded above by,*

$$D^* \leq \frac{\alpha^{-1}(\beta + \Delta)\{1 - (1-\alpha)^{H^*}\}}{(1-\alpha)^{H^*-1}}. \tag{3.14}$$

Comparing with a network of FIFO schedulers, we see that in a network of SETF(0) schedulers, the network utilization level can be kept as high (i.e., as close to 1) as wanted: *unlike FIFO, there is no limit on the maximum allowable network utilization level.* However, since the worst-case edge-to-edge delay bound is inversely proportional to $(1-\alpha)^{H^*-1}$, it increases exponentially as $\alpha \to 1$. Figure 3.5 compares the worst-case edge-to-edge bounds for a FIFO network and an SETF(0) network (with $H^* = 8$) as a function of the network utilization level α. In this example we assume that the capacity of all links is

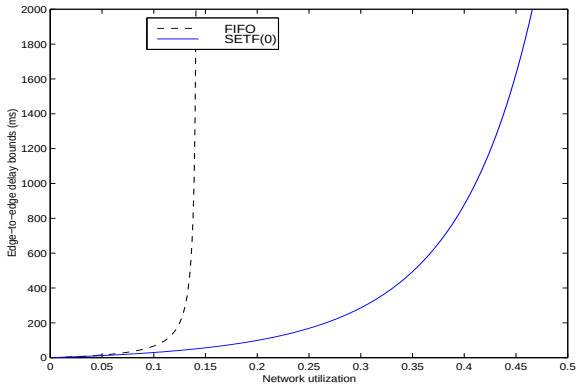


Figure 3.5: Performance comparison: SETF(0) vs. FIFO.

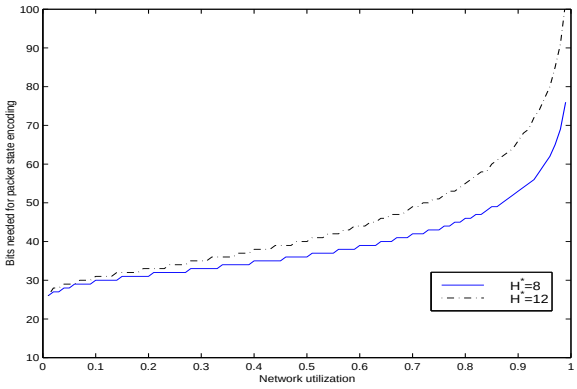


Figure 3.6: No. of bits needed for encoding for SETF(0).

$10\,Gb/s$, and all packets have the same size $L = 1000$ bytes. We set the *network burstiness factor* β in a similar manner as in [11]: we assume that the token bucket size of each flow is bounded in such a way that $\sigma^j \leq \beta_0 \rho^j$, where β_0 (measured in units of time) is a constant for all flows. For a given network utilization level α, we then set $\beta = \alpha\beta_0$. In all the numerical studies presented in this chapter, we choose $\beta_0 = 25\,ms$. From Figure 3.5, it is clear that for a given network utilization level, the worst-case edge-to-edge delay bound for an SETF(0) network is much better than that for a FIFO network[4].

3.4.1.2 Time Stamp Encoding and Performance Trade-offs

In this section we discuss the implication of the worst-case edge-to-edge delay bound on the number of bits needed to encode the time stamp information. Suppose that C^* is the maximum link capacity of the network. Then it is sufficient to have a time granularity of $\iota = 1/C^*$ to mark the precise time each bit of data enters the network[5]. In other words,

[4]When the network utilization is extremely low (below 3% in this example), the worst-case edge-to-edge delay bound of a FIFO network is tighter than that of an SETF(0) network with the same configuration. This is because that, under such an extremely low network utilization, the term Δ in (3.14) becomes a dominant factor in the worst-case edge-to-edge delay bound of an SETF(0) network but has no effect on that of a FIFO network. This can be shown more formally as follows: when α is small, we can approximate the bound of D^* of an SETF(0) network as,

$$D^* \leq \frac{\alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{H^*}\}}{(1-\alpha)^{H^*-1}} \approx \frac{(\beta+\Delta)H^*}{1-(H^*-1)\alpha},$$

which is greater than the worst-case edge-to-edge delay bound of a FIFO network with the same configuration. However, this phenomineon should not have significant importance because our interest is on higher network utilizations.

[5]Although theoretically speaking the finest time granularity $\Gamma = 0$, it is obvious that in practice $\iota = 1/C^*$ is sufficient, as no two bits can arrive at any link within ι units of time.

$\iota = 1/C^*$ is the finest time granularity needed to represent packet time stamps. In the remainder of this chapter we will assume that the clock granularity of the edge devices that place time stamps on packets entering the network is at least ι, i.e., the clocks tick (at least) every ι units of time. We now investigate the problem of how many bits are needed to encode the packet time stamps.

Suppose that m bits are sufficient to encode the packet time stamps precisely. Then the time-stamp bit string wraps around every $2^m\iota$ units of time. Given that the worst-case edge-to-edge delay of a packet in the network is bounded above by D^*, we must have $2D^* \leq 2^m\iota$ so as to enable any SETF(0) scheduler to correctly distinguish and compare the time stamps of two different packets[6]. From Theorem 2, we have

$$m \geq \log_2\{\frac{\alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{H^*}\}}{((1-\alpha)^{H^*-1})\iota}\} + 1. \tag{3.15}$$

Figure 3.6 shows the number of bits needed for packet time stamp encoding for two SETF(0) networks with $H^* = 8$ and $H^* = 12$, respectively. The other parameters used in this example are the same as in Figure 3.5. In particular, $C^* = 10Gb/s$, and thus $\iota = 1/C^* = 10^{-7}\,ms$. As expected, the number of bits needed for packet time stamp encoding increases as the network utilization level increases; it also increases as the network diameter scales up. From this figure we also see that even for a relative low network utilization level, the number of bits required for packet time stamp encoding is relatively large. For example, with $H^* = 8$, 26 bits are needed for $\alpha = 0.1$. Consequently, to achieve a meaningful network utilization level, an SETF(0) network requires a large number of bits for packet time stamp encoding, thus incurring significant control overhead. In the following section, we will show how this problem can be potentially addressed by using coarser time granularity for packet time stamp encoding.

3.4.2 SETF with Coarser Time Granularity: SETF(Γ)

In this section we analyze the SETF(Γ) packet scheduling algorithm with coarser time granularity, i.e., $\Gamma > 0$, and illustrate how the time granularity affects the performance

[6]Here we assume that no extra clock or other timing device/mechanism is used to assist an SETF(0) scheduler to distinguish the packet time stamps. In other words, an SETF(0) scheduler must use the bit strings encoded in the packet header to determine whether the time stamp of one packet is smaller than that of another packet. This can be achieved, for example, by using the *lollipop sequence number* technique [49]. Note that if we assume that each SETF(0) scheduler has a clock that is synchronized with the edge time stamping devices, and thus can use this clock to identify the time slot the current time corresponds to, then it is sufficient to have $2^m\iota \geq D^*$, i.e., one less bit is needed in this case.

trade-offs of an SETF network. In particular, we demonstrate that using a coarser time granularity can potentially reduce the number of bits needed to encode the packet time stamps, albeit at the expenses of sacrificing the maximum allowable network utilization.

Consider a network of SETF(Γ) schedulers. Recall that under SETF(Γ), the time is divided into time slots and packets released into the network during the same time slot carry the same time stamp value (i.e., the time slot number). Clearly the coarser the time granularity Γ is, more packets will be time-stamped with the same time slot number. In particular, if Γ is larger than the worst-case edge-to-edge delay of the network, then a network of SETF(Γ) schedulers degenerates to a network of FIFO schedulers. In the following we will employ the same approach as before to derive performance bounds for a network of SETF(Γ) schedulers.

We first introduce a new notation, h^*: for a given Γ, define $h^* + 1$ to be *the minimum number of hops that any packet can reach within* Γ *units of time after it is released into the network*. Mathematically, h^* is the smallest h such that the following relation holds for all packets:

$$\min_{\text{all } p\text{'s}} \{a^p_{h^*+1} - a^p_1\} \geq \Gamma. \tag{3.16}$$

Note that if $h^* = 0$, we must have $\Gamma = 0$. This gives us SETF(0). On the other hand, if Γ is large enough such that $h^* = H^* - 1$, SETF(Γ) becomes FIFO. Hence, without loss of generality, in the rest of this section we assume that $1 \leq h^* < H^* - 1$. Given this definition of h^*, we have the following bound on $Q(a^p_S)$, where the notations used in the lemma are defined as before:

Lemma 5 *Consider an SETF(Γ) scheduler S with capacity C. Suppose S is the ith hop on the path of a packet p. Then*

$$Q(a^p_S) \leq \alpha C\tau^* + \beta C \quad \textit{if } 1 \leq i \leq h^*, \tag{3.17}$$

and

$$Q(a^p_S) \leq \alpha C\{\tau^* - (a^p_S - a^p_{h^*+1})\} + \beta C + L^{max} \quad \textit{if } h^* < i \leq h \tag{3.18}$$

where recall that $a^p_{h^*+1}$ is the time packet p reaches its ($h^* + 1$)th hop on its path.

The results in Lemma 5 are intuitively clear: similar to SETF(0), (3.18) holds because by the definition of h^*, a packet entering the network in the time interval $[a^p_{h^*+1}, a^p_S]$ must have a time stamp that is larger than that of packet p. On the other hand, for $i \leq h^*$, packets entering the network after a^p_1 but before a^p_S may have the same time stamp as packet p, i.e., they may carry the same time slot number. Hence the same bound as the FIFO bound applies in (3.17).

Using Lemma 5, we can derive the following recursive relations: for $i = 1, \ldots, h^*$,

$$a^p_{i+1} = f^p_i = a^p_i + \alpha\tau^* + \beta.$$

and for $i = h^* + 1, \ldots, h$ (for simplicity, define $a^p_{h+1} = f^p_h$),

$$a^p_{i+1} = f^p_i = (1-\alpha)a^p_i + \alpha(\tau^* + a^p_{h^*+1}) + \beta + \Delta.$$

Solving these recursive equations, we have

Lemma 6 *Consider a packet p which traverses a path with h hops. Then for $i = 1, 2, \ldots, h^*$,*

$$f^p_i - a^p_1 \leq i(\alpha\tau^* + \beta); \tag{3.19}$$

and for $i = h^* + 1, \ldots, h$,

$$f^p_i - a^p_1 \leq h^*(\alpha\tau^* + \beta) + \tau^*\{1 - (1-\alpha)^{i-h^*}\} + \alpha^{-1}(\beta + \Delta)\{1 - (1-\alpha)^{i-h^*}\}. \tag{3.20}$$

Applying Lemma 6, we obtain the following performance bounds for a network of SETF(Γ) schedulers.

Theorem 3 *Consider a network of SETF(Γ) schedulers with a network diameter H^*. If the network utilization level α satisfies the following condition,*

$$(1-\alpha)^{H^*-h^*-1} > \alpha h^*, \tag{3.21}$$

then

$$\tau^* \leq \frac{\beta h^* + \alpha^{-1}(\beta + \Delta)\{1 - (1-\alpha)^{H^*-h^*-1}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*}. \tag{3.22}$$

Furthermore, the worst-case edge-to-edge delay is bounded above by,

$$D^* \leq \frac{\beta h^* + \alpha^{-1}(\beta + \Delta)\{1 - (1-\alpha)^{H^*-h^*}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*}. \tag{3.23}$$

Note first that in Theorem 3, setting $h^* = 0$ yields the results for a network of SETF(0) schedulers, whereas setting $h^* = H^* - 1$ yields the results for a network of FIFO schedulers (with a difference of $\frac{\Delta}{1-(H^*-1)\alpha}$ caused by the extra care taken by the analysis of an SETF network to accout for the non-preemptive property of an SETF scheduler). Hence Theorem 2 and Theorem 1 can be considered as two special cases of Theorem 3. In general, Theorem 3 states that with a coarser time granularity $\Gamma > 0$ (which determines h^*), we can no longer set the network utilization level at any arbitrary level, as in the case of SETF(0), while still having a *finite* worst-case edge-to-edge delay bound. In other words, for a given $\Gamma > 0$, there is a limit on the maximum allowable network utilization level as imposed by the condition (3.21). This limit on the maximum allowable network utilization level is the performance penalty we pay for using coarser time granularity to represent the packet time stamp information.

3.4.2.1 Time Stamp Encoding and Performance Trade-offs

In this section we show that using coarser time granularity can potentially reduce the number of bits needed for packet time stamp encoding. We also illustrate through numerical examples how time granularity affects the performance trade-offs of SETF(Γ) networks.

We first consider the problem of packet time stamp encoding. Using the same argument as in Section 3.4.1.2, for a given time granularity Γ and network utilization level α, the number of bits m needed for packet time stamp encoding must satisfy the following condition:

$$m \geq \log_2\{\frac{\beta h^* + \alpha^{-1}(\beta + \Delta)\{1 - (1-\alpha)^{H^*-h^*}\}}{((1-\alpha)^{H^*-h^*-1} - \alpha h^*)\Gamma}\} + 1. \tag{3.24}$$

From (3.24), we see that for a fixed network utilization level α, larger Γ *may* reduce the number of bits needed for packet time stamp encoding. *However*, as we increase Γ, h^* may also be increased. Consequently, the right hand side of (3.24) may increase. Hence the relationship between m and Γ is *not strictly monotone*. Furthermore, a larger Γ in general also yields a smaller maximum allowable network utilization level bound. From Theorem 3, (3.24) and the definition of h^* (3.16), it is not too hard to see that given a network with

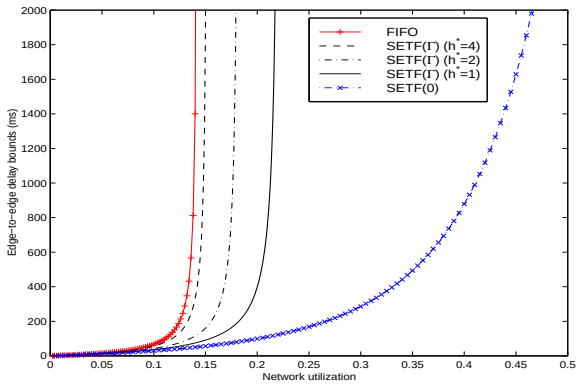


Figure 3.7: Performance comparison: SETF vs. FIFO.

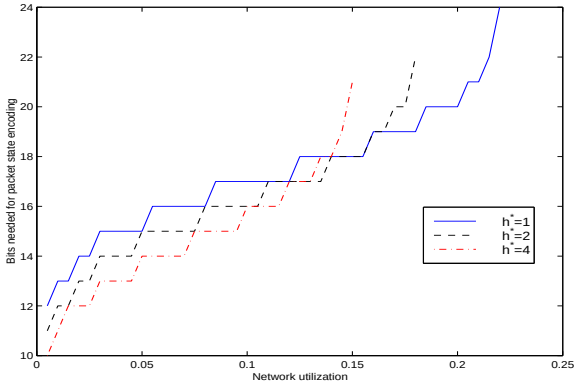


Figure 3.8: No. of bits needed for encoding for SETF(Γ).

diameter H^*, we can essentially divide the time granularity Γ into H^* *granularity levels*: each granularity level corresponds to one value of $h^* = 0, 1, \ldots, H^* - 1$. The *finest* granularity level corresponds to $h^* = 0$, and the *coarsest* granularity level to $h^* = H^* - 1$. For this reason, in the following numerical studies, we will use h^* to indicate the time granularity used in an SETF(Γ) network. These numerical studies demonstrate the performance trade-offs involved in the design of SETF(Γ) networks using coarser time granularity, i.e., when the number of bits for packet time stamp encoding is limited. In all these studies, except for the network diameter H^* all other system parameters (link capacity, packet size, β) are the same as used in Figure 3.5.

Figure 3.7 shows the effect of time granularity on the worst-case edge-to-edge delay bound for an SETF(Γ) network with $H^* = 8$. For comparison, we also include the results for the corresponding FIFO network. From the figure it is clear that coarser time granularity (i.e., larger h^*) yields poorer worst-case edge-to-edge delay bound. As the time granularity gets coarser (i.e., h^* increases), the worst-case edge-to-edge delay bound quickly approaches to that of the FIFO network.

Next we illustrate how the network utilization level of an SETF(Γ) network affects the number of bits needed for packet time stamp encoding. Figure 3.8 shows the number of bits needed for packet time stamp encoding as a function of the network utilization level under various time granularities (as indicated by h^*). In this example, the network diameter $H^* = 8$. From this figure we see that for low network utilization levels, using coarser time granularity reduces the number of bits needed for packet time stamp encoding. However, as coarser time granularity also imposes a tight bound on the maximum allowable network utilization, this reduction in the number of bits needed for packet time stamp encoding

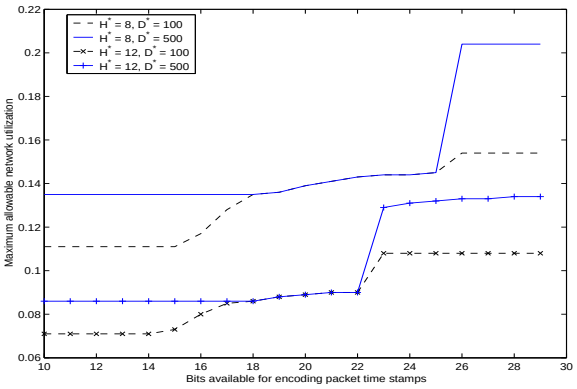


Figure 3.9: No. of bits for encoding, network diameter, and maximum allowable network utilization.

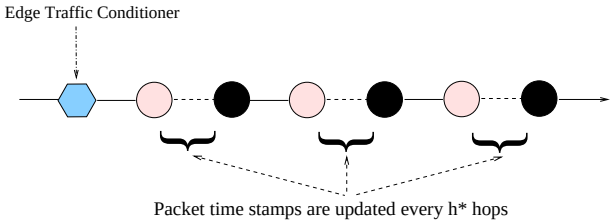


Figure 3.10: Updating packet time stamps inside the network core.

may not be feasible when the network utilization level is increased (this is why the curve for a given time granularity (h^*) stops at certain network utilization level). To put it in another way, to achieve a higher network utilization level, SETF(Γ) schedulers with *finer* time granularity (thus smaller h^*) must be used, thus requiring more bits for packet time stamp encoding.

In the last set of numerical studies, we demonstrate how the number of bits available for packet time stamp encoding affects the maximum allowable network utilization so as to support a given target worst-case edge-to-edge delay bound for SETF networks. The results are shown in Figure 3.9, where networks with a combination of the network diameters $H^* = 8$ and $H^* = 12$ and delay bounds $D^* = 100\,ms$ and $D^* = 500\,ms$ are used. As we can see from the figure that for a given number of bits for packet time stamp encoding, as the network diameter increases, the maximum allowable network utilization decreases. Note also that when the number of bits for packet time stamp encoding is small (e.g., less than 15 for a network with parameters $H^* = 8$ and $D^* = 100\,ms$), the packet time stamp does no enhance the performance of a SETF(Γ, h^*) network, and the SETF(Γ, h^*) network behaves essentially as a FIFO network with a maximum network utilization level around 0.11. Beyond this threshold, as the number of bits used increases, the maximum allowable network utilization also increases. However, as the figure shows, further increasing the number of bits beyond a certain value (e.g., 26 for a network with parameters $H^* = 8$ and $D^* = 100\,ms$) for encoding will not improve the maximum allowable network utilization.

3.5 Network of Dynamic Earliest Time First Schedulers

So far we have seen that by including additional control information in the packet header and adding sophistication/complexity at network schedulers, the class of SETF packet scheduling algorithms improve upon the maximum allowable network utilization and worst-case edge-to-edge delay bounds of the simple FIFO packet scheduling algorithm. This performance improvement comes essentially from the ability of an SETF scheduler to limit the effect of "newer" packets on "older" packets. However, the provisioning power of SETF packet scheduling algorithms is still rather limited. Given the finest time granularity to encode the packet time stamps, although we can achieve arbitrary network utilization in a network of SETF(0) schedulers, the worst-case edge-to-edge delay bound is inversely proportional to $(1-\alpha)^{H^*}$. Hence the bound grows exponentially, as the network diameter H^* increases. In addition, with coarser time granularities, the performance of SETF networks deteriorates further. In this section we devise another class of aggregate packet scheduling algorithms—the class of DETF algorithms—which with further "sophistication/complexity" added at the schedulers, achieve far superior performance.

In the general definition of a DETF packet scheduling algorithm, we use two parameters: the time granularity Γ and the *(packet) time stamp increment hop count* h^*. Note that unlike SETF where h^* is determined by Γ, here h^* is *independent of* Γ. Hence we denote a DETF scheduler by DETF(Γ, h^*). In the following, we will present the definition of DETF($0, h^*$) first, i.e., DETF with the finest time granularity. The general definition of DETF(Γ, h^*) will be given afterwards.

As in the case of SETF(0), the time stamp of a packet in a network of DETF($0, h^*$) schedulers is represented precisely. In particular, it is initialized at the network edge with the time the packet is released into the network. Unlike SETF(0), however, the time stamp of the packet will be updated every h^* hops (see Figure 3.10). Formally, suppose packet p traverses a path of h hops. Let ω_0^p denote the time stamp of packet p as it is released into the network, i.e., $\omega_0^p = a_1^p$. Let $\kappa = \lceil \frac{h}{h^*} \rceil$. For $k = 1, 2, \ldots, \kappa - 1$, the time stamp of packet p is updated *after* it has traversed the kh^*th hop on its path (or as it enters the $(kh^* + 1)$th hop on its path). Let ω_k^p denote the packet time stamp of packet p after its kth update. The packet time stamp ω_k^p is updated using the following *update rule*:

$$\omega_k^p := \omega_{k-1}^p + d^*, \qquad k = 1, \ldots, \kappa - 1, \tag{3.25}$$

where the parameter $d^* > 0$ is referred as the *(packet) time stamp increment*. *We impose*

the following condition on d^* that relates the packet time stamp ω_k^p to the actual time packet p departs the kh^*th hop:

$$\text{for } k = 1, \ldots, \kappa - 1, \quad f_{kh^*}^p \leq \omega_k^p, \text{ and } f_h^p \leq \omega_\kappa^p := \omega_{\kappa-1}^p + d^*. \tag{3.26}$$

This condition on d^* is referred to as the *reality check* condition. Intuitively, we can think of the path of packet p being partitioned into κ *segments* of h^* hops each (except for the last segment, which may be shorter than h^* hops). The reality check condition (3.26) ensures that the packet time stamp carried by packet p after it has traversed k segments is not smaller that the actual time it takes to traverse those segments. In the next section we will see that the reality check condition (3.26) and the packet time stamp update rule (3.25) are essential in establishing the performance bounds for a network of DETF schedulers.

We now present the definition for the general DETF(Γ, h^*) packet scheduling algorithm with a (coarser) time granularity $\Gamma > 0$. As in the case of SETF(Γ), in a network of DETF(Γ, h^*) schedulers, the time is divided into time slots of Γ units: $[(n-1)\Gamma, n\Gamma), n = 1, 2, \ldots$, and all packet time stamps are represented using the time slots. In particular, if packet p is released into the network in the time slot $[(n-1)\Gamma, n\Gamma)$, then $\omega_0^p = n\Gamma$. We also require that *the packet time stamp increment d^* be a multiple of* Γ. Hence the packet time stamp ω_k^p is always a multiple of Γ. In practice, we can encode ω_k^p as the corresponding time slot number (as in the case of SETF(Γ)).

3.5.1 Performance Bounds for a Network of DETF Schedulers

In this section we establish performance bounds for a network of DETF schedulers. In particular, we will show that by using *dynamic* packet time stamps, we can obtain significantly better performance bounds for a network of DETF schedulers than those for a network of SETF schedulers.

Consider a network of DETF(Γ, h^*) schedulers, where $\Gamma \geq 0$ and $1 \leq h^* \leq H^*$. We first establish an important lemma which bounds *the amount of traffic carried by packets at a DETF(Γ, h^*) scheduler whose time stamp values fall within a given time interval*. Consider a DETF(Γ, h^*) scheduler S. Given a time interval $[\tau, t]$, let $\mathcal{M}$ be the set of packets that traverse S at some time whose time stamp values fall within $[\tau, t]$. Namely, $p \in \mathcal{M}$ if and only if for some $k = 1, 2, \ldots, \kappa,$, S is on the kth segment of packet p's path, and $\tau \leq \omega_{k-1}^p \leq t$. For any $p \in \mathcal{M}$, we say that packet p *virtually* arrives at S during $[\tau, t]$.

Let $\tilde{A}_S(\tau, t)$ denote the total amount of traffic virtually arriving at S during $[\tau, t]$, i.e., total amount of traffic carried by packets in $\mathcal{M}$. Then we have the following bound on $\tilde{A}_S(\tau, t)$.

Lemma 7 *Consider an arbitrary scheduler S with capacity C in a network of DETF(Γ, h^*) schedulers. For any time interval $[\tau, t]$, let $\tilde{A}(\tau, t)$ be defined as above. Then*

$$\tilde{A}(\tau, t) \leq \beta C + \alpha C(t - \tau + \Gamma). \tag{3.27}$$

Proof: For simplicity, we first prove a bound on $\tilde{A}^j(\tau, t)$, the amount of traffic virtually arriving at S during $[\tau, t]$ from a flow j. Consider an arbitrary packet p of flow j which virtually arrives at S (on the kth segment) during $[\tau, t]$, i.e., $\tau \leq \omega_{k-1}^p \leq t$. From (3.25), it is easy to see that,

$$\omega_{k-1}^p = \omega_0^p + (k-1)d^*.$$

Becase $\tau \leq \omega_{k-1}^p \leq t$, we have,

$$\tau - (k-1)d^* \leq \omega_0^p \leq t - (k-1)d^*.$$

Therefore,

$$\tilde{A}^j(\tau, t) \leq \sigma^j + \rho^j \lceil \frac{t - (k-1)d^* - (\tau - (k-1)d^*)}{\Gamma} \rceil \Gamma \leq \sigma^j + \rho^j(t - \tau + \Gamma). \tag{3.28}$$

From (3.28) and the edge traffic provisioning condition (3.2), the lemma follows easily. ∎

Note that if $\Gamma = 0$, the bound on $\tilde{A}(\tau, t)$ is exactly the same as the edge traffic provisioning condition (3.2). Intuitively, (3.27) means that using the (dynamic) packet time stamp with the finest time granularity, the amount of traffic *virtually* arriving at S during $[\tau, t]$ is bounded in a manner as if the traffic were re-shaped at S using (3.2). In the general case where a coarser time granularity $\Gamma > 0$ is used, an extra $\alpha C\Gamma$ amount of traffic may (virtually) arrive at S, as opposed to (3.2) at the network edge. This is not surprising, since with a coarser time granularity, a scheduler S inside the network core cannot distinguish a packet from those other packets that traverse S and have the same time stamp value.

From Lemma 7, we can derive a recursive relation for ω_k^p's using a similar argument as used before. Based on this recursive relation, we can establish performance bounds for a network of DETF(Γ, h^*) schedulers. The general results are somewhat "messy" to state. For brevity, in the following we present results for three special but *representative* cases. As we will see later, the first two theorems are sufficient to demonstrate the provisioning power of a network of DETF schedulers. The third theorem is included here for comparison purpose.

Theorem 4 (A Network of DETF(0,1) Schedulers) *Consider a network of DETF(0,1) schedulers with a network diameter H^*. Let $d^* = \beta + \Delta$, then the reality condition (3.26) holds. Furthermore, for any $0 < \alpha < 1$, the worst-case edge-to-edge delay D^* is bounded above by $D^* \leq H^* d^* = H^*(\beta + \Delta)$.*

Theorem 5 (A Network of DETF($\Gamma, 1$) Schedulers) *Consider a network of DETF($\Gamma, 1$) schedulers with a network diameter H^*, where $\Gamma > 0$. Let $d^* = \lceil (\alpha\Gamma + \beta + \Delta)/\Gamma \rceil \Gamma$, then the reality condition (3.26) holds. Furthermore, for any $0 < \alpha < 1$, the worst-case edge-to-edge delay D^* is bounded above by $D^* \leq H^* d^* + \Gamma$.*

Theorem 6 (A Network of DETF(Γ, h^*) Schedulers with $d^* = \Gamma$) *Consider a network of DETF(Γ, h^*) schedulers with a network diameter H^*, where $\Gamma > 0$ (and $h^* > 1$). We set $d^* = \Gamma$,* i.e., the packet time stamp is advanced exactly one time slot every time it is updated. *Let $\kappa^* = \lceil \frac{H^*}{h^*} \rceil$. Suppose the network utilization level α and the time granularity Γ satisfy the following condition:*

$$0 < \frac{h^*(\alpha\Gamma + \beta + \Delta)}{1 - (h^* - 1)\alpha} \leq d^* = \Gamma. \tag{3.29}$$

Then the worst-case edge-to-edge delay D^* is bounded above by $D^* \leq (\kappa^* + 1)\Gamma$.

Note that a DETF(0,1) scheduler is a special case of the Virtual-Time Earliest-Deadline-First (VT-EDF) packet scheduling algorithm proposed in [86] under the *virtual time reference system* framework, where the delay parameter for all flows is set to d^*. In general, regarding the per-hop scheduling behavior, DETF is close to a special case of SCED+ by Cruz [21]. However, SCED+ only considers discrete time and does not study the effect of number of bits available for packet state encoding on the performance of a network.

From Theorem 4 and Theorem 5, we see that with $h^* = 1$, the worst-case edge-to-edge delay bound is linear in the network diameter H^*. Furthermore, with the finest time granularity, the worst-case edge-to-edge delay bound is *independent of the network utilization level* α. This is because the *per-hop delay* is bounded by $d^* = \beta + \Delta$. With a coarser time granularity $\Gamma > 0$, *per-hop delay* is bounded by $d^* = \lceil(\alpha\Gamma + \beta + \Delta)/\Gamma\rceil\Gamma$, where the network utilization level determines the "additional delay" ($\alpha\Gamma$) that a packet may experience at each hop.

From Theorem 6, we see that in a network of DETF(Γ, h^*) where $d^* = \Gamma$ and $h^* > 1$, *the maximum allowable network utilization is bounded*. To see why this is the case, first note that we must have $\alpha < 1/(h^* - 1)$, otherwise the left hand side of (3.29) becomes infinitely. For a given $\Gamma > 0$, the condition (3.29) imposes the following tighter bound on α:

$$\alpha < \frac{1 - h^*(\beta + \Delta)\Gamma^{-1}}{2h^* - 1} < \frac{1}{2h^* - 1} < \frac{1}{h^* - 1}. \tag{3.30}$$

For a given α that satisfies (3.30), comparing the worst-case edge-to-edge delay bound in Theorem 6 to that of a network of FIFO schedulers with a network diameter h^*, we see that updating packet time stamps every h^* hops effectively reduces a network of diameter H^* into a number of smaller networks with diameter h^*. In particular, setting $d^* = \Gamma$ allows us to consider these smaller networks as networks of FIFO schedulers with diameter h^*. By appropriately taking into account the effect of dynamic time stamps with coarser time granularity (the extra $\alpha\Gamma + \Delta$ factor), Theorem 6 can essentially be obtained from the bound for a network of FIFO schedulers.

3.5.2 Packet State Encoding

In this section we first consider the problem of packet state encoding for a network of DETF schedulers, namely, the number of bits that is needed to encode the *dynamic* packet time stamp *and* possibly other control information for the *proper operation* of a DETF network.

First consider a network of DETF(0,1) schedulers with a network diameter H^*. As in the case of SETF(0), we use ι to denote the finest time granularity necessary to represent the packet time stamps, i.e., $\iota = 1/C^*$, where C^* is the maximum link capacity of the network. From Theorem 4, we see that the number of bits m that is needed to encode the (dynamic) packet time stamps precisely must satisfy the following condition:

$$2^{m-1}\iota \geq H^*(\beta + \Delta), \text{ or } m \geq \log_2 H^* + \log_2[(\beta + \Delta)/\iota] + 1. \tag{3.31}$$

Now consider a network of DETF($\Gamma, 1$) with a coarser time granularity $\Gamma > 0$. From Theorem 5, for a given network utilization level α, we see that the number of bits m that is needed to encode the (dynamic) packet time stamps must satisfy the following condition:

$$2^{m-1}\Gamma \geq H^*\lceil\frac{\alpha\Gamma+\beta+\Delta}{\Gamma}\rceil\Gamma+\Gamma, \text{ or } m \geq \log_2\{H^*\lceil\frac{\alpha\Gamma+\beta+\Delta}{\Gamma}\rceil+1\}+1. \quad (3.32)$$

Hence for a given network utilization level α, coarser time granularity (i.e., larger Γ) in general leads to fewer bits needed to encode the dynamic packet time stamps. However, due to the ceiling operation in (3.32), at least $\log_2\{H^*+1\}+1$ bits are needed. This *effectively* places a bound on the range of time granularities that should be used, i.e., $\Gamma \in [0, (\beta+\Delta)/(1-\alpha)]$. Any coarser time granularity $\Gamma > (\beta+\Delta)/(1-\alpha)$ will not reduce the minimum number of bits, $\log_2\{H^*+1\}+1$, needed for packet time stamp encoding.

In the general case where $h^* > 1$, in order to ensure a DETF(Γ, h^*) scheduler to work properly, not only do we need to encode the packet time stamps, we also need *some additional control information* to be carried in the packet header of each packet: in order for a scheduler to know whether the packet time stamp of a packet must be updated, we include a *hop-count counter* as part of the packet state carried in the packet header to record the number of hops a packet has traversed. This hop-count counter is incremented every time a packet traverses a scheduler, and it is reset when it reaches h^*. Thus the hop-count counter can be encoded using $\log_2 h^*$ number of bits. Therefore for a network of DETF(Γ, h^*) where d^* is set to Γ, from Theorem 6 the total number of bits needed for packet state encoding is given by

$$m \geq \log_2\{\kappa^*+1\}+1+\log_2 h^*, \quad (3.33)$$

provided that the network utilization level α and the time granularity Γ are chosen in such a manner that (3.29) holds. Note that from (3.33) we have $m \geq \log_2\{\kappa^* h^*+h^*\}+1 \geq \log_2\{H^*+h^*\}+1$. Therefore, the number of bits needed for encoding the packet states is increased as h^* increases. Moreover, via (3.29) h^* also affects the maximum allowable network utilization bound. In particular, from (3.30) a larger h^* leads to a smaller bound on the maximum allowable network utilization. For these reasons it is sufficient to only consider networks of DETF($\Gamma, 1$) schedulers[7].

[7]In practice, it is possible to implement the hop-count counter using, say, the TTL field in the IP header, thus avoiding the extra $\log_2 h^*$ bits. For example, we have implemented two versions of DETF packet scheduling algorithms in FreeBSD: one using IP-IP tunneling technique, where the TTL field in the en-

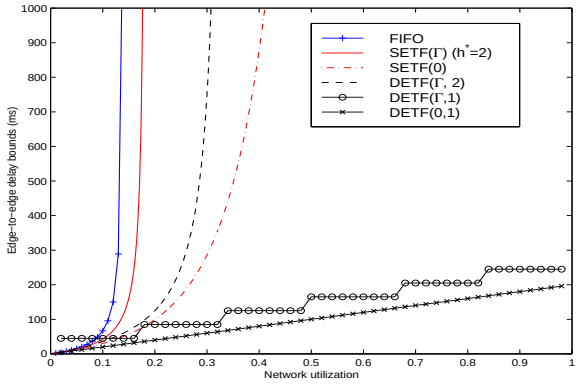


Figure 3.11: Edge-to-edge delay bound comparison ($H^* = 8$).

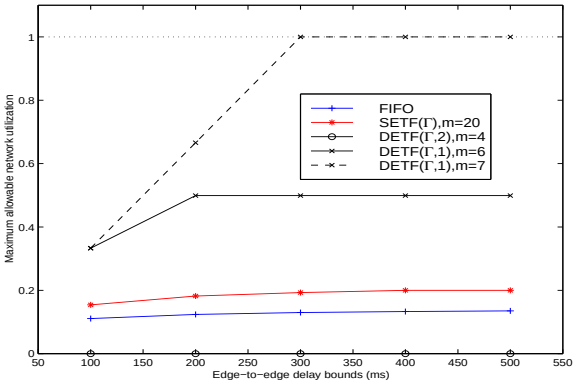


Figure 3.12: Provisioning power of FIFO, SETF(Γ), DETF($\Gamma, 1$), and DETF($\Gamma, 2$) networks ($H^* = 8$).

3.5.3 Performance Trade-offs and Provisioning Power

In this section we use numerical examples to demonstrate the performance trade-offs in the design of DETF networks. By comparing the performance of FIFO, SETF and DETF networks, we also illustrate the provisioning power of the aggregate packet scheduling algorithms in support of guaranteed delay service. Lastly, we briefly touch on the issue of complexity/cost in implementing the aggregate packet scheduling algorithms.

The network setting for all the studies is the same as before. Namely, all links have a capacity of $10\,Gb/s$, all packets have a size of $L = 1000\,B$, and $\beta = \alpha\beta_0$, where α is the network utilization level and $\beta_0 = 25\,ms$. The network diameter H^* and the network utilization level α will be varied in different studies.

In the first set of numerical examples, we illustrate the relationship between the network utilization level α and the worst-case edge-to-edge delay bound for networks employing various aggregate packet scheduling algorithms. The results are shown in Figure 3.11, where $H^* = 8$ is used for all the networks. For the SETF(Γ) network, we choose $\Gamma = 2\Delta = 0.8\mu s$ (i.e., $h^* = 2$). Whereas in the DETF($\Gamma, 2$) network, the time granularity Γ is chosen in such a way that (3.29) in Theorem 6 holds. For the DETF($\Gamma, 1$) network, we set $\Gamma = 5\,ms$. From the figure we see that the DETF(0,1) network has the best worst-case edge-to-edge delay bound. Despite a relatively coarser time granularity, the delay bound

capsulating IP header is used to record the hop count a packet has traversed within a given network domain and serves as the hop-count counter; another using MPLS, where the TTL field in the MPLS label is used for the same purpose. In both cases, we only need additional bits to encode the packet time stamps. In such situations, a network of DETF(Γ, h^*) schedulers with $d^* = \Gamma > 0$ and $h^* > 1$ requires only $\log_2 \kappa^* + 1$ additional number of bits.

for the DETF($\Gamma, 1$) network is fairly close to that of the DETF(0,1) network. In addition, when the network utilization level is larger than 0.2, the DETF($\Gamma, 1$) network also has a better delay bound than the rest of the networks. From Theorem 5, it is clear that the worst-case edge-to-edge delay bound for a DETF($\Gamma, 1$) network decreases (and approaches to that of a DETF($0, 1$) network), when finer time granularity (smaller Γ) is used. The delay bound of the DETF($\Gamma, 2$) network is worse than that of the SETF(0) network (with the finest time granularity), but is considerably better than those of the SETF(Γ) and FIFO networks. From this example, we see that the DETF networks in general have far better delay performance than those of SETF and FIFO networks.

In the next set of numerical examples, we compare the provisioning power of the various aggregate packet scheduling algorithms. In particular, we consider the following *provisioning* problem: given a network employing a certain aggregate packet scheduling algorithm, what is the maximum allowable network utilization level we can attain in order to meet a *target* worst-case edge-to-edge delay bound? In this study, we allow networks employing different aggregate packet scheduling algorithms to use different number bits for packet state encoding. More specifically, the FIFO network needs no additional bits. The SETF(Γ) network (where Γ is chosen such that $h^* = 1$) uses 20 additional bits for time stamp encoding. The number of additional bits used by the DETF($\Gamma, 2$) network is 3. For the DETF($\Gamma, 1$) networks, we consider two cases: one uses 6 additional bits, while the other uses 7 bits. All the networks used in these studies have the same diameter $H^* = 8$. Figure 3.12 shows the maximum allowable network utilization level as a function of the target worst-case edge-to-edge delay bound for the various networks. The results clearly demonstrate the performance advantage of the DETF networks. In particular, with a few number of bits needed for packet state encoding, the DETF($\Gamma, 1$) networks can attain much higher network utilization level, while supporting the same worst-case edge-to-edge delay bound.

In the last set of numerical examples, we focus on the DETF($\Gamma, 1$) networks only. In this study, we investigate the design and performance trade-offs in employing DETF($\Gamma, 1$) networks to support guaranteed delay service. In particular, we consider the following problem: given a *fixed* number of bits for packet state encoding, what is the maximum allowable network utilization level that we can attain to support a target worst-case edge-to-edge delay bound? Note that for a network of diameter H^*, at least $\log_2\{H^*+1\}+1$ bits are needed for packet state encoding. More bits available will allow us to choose finer time granularity for time stamp encoding, thus yielding a better delay bound as well as a higher maximum

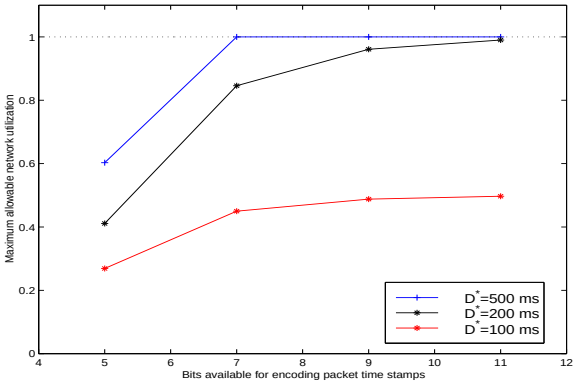


Figure 3.13: Design and performance trade-offs for DETF($\Gamma, 1$) networks ($H^* = 8$).

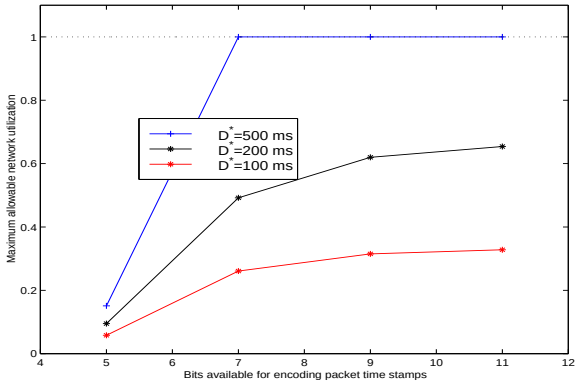


Figure 3.14: Design and performance trade-offs for DETF($\Gamma, 1$) networks ($H^* = 12$).

network utilization level. In Figure 3.13 we show, for a network of diameter $H^* = 8$, how the number of bits available for packet state encoding affects the maximum network utilization level so as to support a given target worst-case edge-to-edge delay bound. The same results for a network of diameter $H^* = 12$ are shown in Figure 3.14. From these results we see that with relatively a few number of bits, a DETF network can achieve fairly decent or good network utilization while meeting the target worst-case edge-to-edge delay bound. In particular, with the target worst-case edge-to-edge delay bounds $200\,ms$ and $500\,ms$, we can achieve more than 50% (and up to 100%) network utilization level using only 6 to 7 additional bits. Comparing Figure 3.13 and Figure 3.14, it is clear that a network with larger diameter requires more bits than a network with smaller diameter to achieve the same maximum allowable network utilization. However, the minimum number of bits required for packet state encoding grows only logarithmically with the network diameter H^*. Furthermore, today's networks tend to be more "dense", i.e., with relative small H^*. Hence with relatively small number of additional bits (e.g., 8 or 16 bits) for time stamp encoding, we can design DETF($\Gamma, 1$) networks to attain fairly high network utilization while supporting reasonably good edge-to-edge delay bounds.

We conclude this section by briefly touching on the issue of cost/complexity in implementing the aggregate packet scheduling algorithms. Besides the fact that additional bits are needed for packet state encoding, both the SETF and DETF packet scheduling algorithms require comparing packet time stamps and sorting packets accordingly. With the finest time granularity, this sorting operation can be expensive. However, with only a few bits used for packet time stamp encoding, sorting can be avoided by implementing a "calendar queue" (or rotating priority queue [54]) with a number of FIFO queues. This particularly favors the

DETF($\Gamma, 1$) packet scheduling algorithms, since the number of bits needed for time stamp encoding can be kept small. However, compared to SETF, DETF($\Gamma, 1$) packet scheduling algorithms require updating packet time stamps at every router, and thus d^* must be configured at each router. Lastly, in terms of finding additional bits for packet state encoding, we can re-use certain bits in the IP header [73]. This is the case in our prototype implementation using the IP-IP tunneling technique, where we re-use the IP identification field (16 bits) in the encapsulating IP header to encode the packet time stamp.

3.6 Summary

In this chapter we investigated the fundamental trade-offs in aggregate packet scheduling for support of (worst-case) guaranteed delay service. Based on a novel analytic approach that focuses on network-wide performance issues, we studied the relationships between the worst-case edge-to-edge delay, the maximum allowable network utilization level and the "sophistication/complexity" of aggregate packet scheduling employed by a network. We designed two new classes of aggregate packet scheduling algorithms—the static earliest time first (SETF) and dynamic earliest time first (DETF) algorithms—both of which employ additional timing information carried in the packet header for packet scheduling, but differ in their manipulation of the packet time stamps. Using the SETF and DETF as well as the simple FIFO packet scheduling algorithms, we demonstrated that with additional control information carried in the packet header and added "sophistication/complexity" at network schedulers, both the maximum allowable network utilization level and the worst-case edge-to-edge delay bound can be significantly improved. We further investigated the impact of the number of bits available for packet state encoding on the performance trade-offs as well as the provisioning power of these aggregate packet scheduling algorithms. In particular, we showed that with relatively small number of bits for packet state encoding, the DETF packet scheduling algorithms can attain fairly good performance bounds. These results illustrate the fundamental trade-offs in the design of aggregate packet scheduling algorithms, and shed light on the provisioning power of aggregate packet scheduling in support of guaranteed delay service.

Chapter 4

Supporting Per-Flow Guaranteed Services

4.1 Introduction

In this chapter we propose and develop a novel *virtual time reference system* as a *unifying* scheduling framework to provide scalable support for guaranteed services. In the same way that the WFQ reference system relates to the IntServ architecture, the proposed virtual time reference system is designed as a *conceptual* framework upon which guaranteed services can be implemented in a scalable manner using the DiffServ paradigm. More specifically, this virtual time reference system provides a unifying framework to characterize, in terms of their abilities to provide delay and bandwidth guarantees, both the *per-hop behaviors* of core routers and the *end-to-end properties* of their concatenation. The key construct in the proposed virtual time reference system is the notion of *packet virtual time stamps*, which, as part of the packet state, are referenced and updated as packets traverse each core router. A crucial property of packet virtual time stamps is that they can be computed using solely the packet state carried by packets (plus a couple of fixed parameters associated with core routers). In this sense, the virtual time reference system is *core stateless*, as no per-flow state is needed at core routers for computing packet virtual time stamps.

In this chapter, we lay the theoretical foundation for the definition and construction of packet virtual time stamps. We describe how per-hop behavior of a core router (or rather its scheduling mechanism) can be characterized via packet virtual time stamps, and based on this characterization, establish end-to-end per-flow delay bounds. Consequently, we

demonstrate that in terms of support for guaranteed services, the proposed virtual time reference system has the same expressive power as the IntServ model. Furthermore, we show that the notion of packet virtual time stamps leads to the design of *new* core stateless scheduling algorithms, especially work-conserving ones. In addition, our framework *does not exclude* the use of existing scheduling algorithms such as *stateful* fair queueing algorithms to support guaranteed services.

The objectives of the proposed virtual time reference system are two-fold. First, as a reference system, it must not *mandate* any specific scheduling algorithms to be employed in a network in order to provide guaranteed services. In other words, it must allow for diverse scheduling algorithms as long as they are capable of providing QoS guarantees. In fact, we will show that our virtual time reference system can accommodate both *core stateless* scheduling algorithms such as CJVC and *stateful* scheduling algorithms. Second, the virtual time reference system provides a QoS abstraction for scheduling mechanisms that *decouples* the data plane from the QoS control plane. This abstraction facilitates the design of a bandwidth broker architecture (either centralized or distributed), where QoS states are maintained *only* at bandwidth brokers, *while still being capable of providing QoS guarantees with similar granularity and flexibility of the IntServ guaranteed service*. We believe that these two objectives are important in implementing guaranteed services in practice. For example, the ability to employ diverse scheduling algorithms not only encourages choice and competition among equipment vendors and Internet service providers (ISPs), but also, perhaps more importantly, allows a network and its services to evolve. Similarly, by maintaining QoS reservation states only in bandwidth brokers, core routers are relieved of QoS control functions such as admission control, making them potentially more efficient. Furthermore, a QoS control plane which is decoupled from the data plane allows an ISP to deploy sophisticated provisioning and admission control algorithms to optimize network utilization without incurring software/hardware upgrades at core routers. This chapter will focus mostly on the theoretical underpinning of the proposed virtual time reference system. We will briefly address the issues regarding its implementation. The problem of designing a bandwidth broker architecture based on the virtual time reference system to support QoS provisioning and admission control will be briefly discussed.

The rest of this chapter is organized as follows. In the next section we will briefly outline the basic architecture of the virtual time reference system. In Section 4.3 we define a virtual time reference system in the context of an ideal per-flow system. This virtual time reference system is extended in Section 4.4 to account for the effect of packet schedul-

ing. Furthermore, end-to-end per-flow delay bounds are also derived using the virtual time reference system. In Section 4.5, we design new core stateless scheduling algorithms using packet virtual time stamps. We then show that existing scheduling algorithms can be accommodated in our framework—simple static scheduling algorithms with resource preconfiguration in Section 4.6 and the generic latency-rate server scheduling framework in Section 4.7. In Section 4.8 we briefly discuss various issues regarding implementation and admission control. This chapter is concluded in Section 4.9.

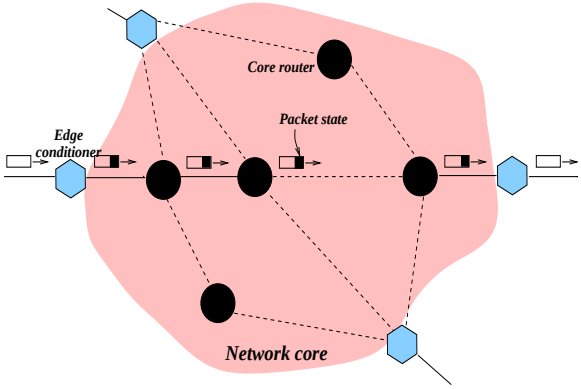
4.2 Virtual Time Reference System: Basic Architecture

In this section we outline the basic architecture of the proposed unifying scheduling framework—the *virtual time reference system*. Conceptually, the virtual time reference system consists of three logical components (see Figure 4.1 and Figure 4.2): *packet state* carried by packets, *edge traffic conditioning* at the network edge, and *per-hop virtual time reference/update mechanism* at core routers. The virtual time reference system is defined and implemented within a *single* administrative domain. In other words, packet state inserted by one administrative domain will not be carried over to another administrative domain.

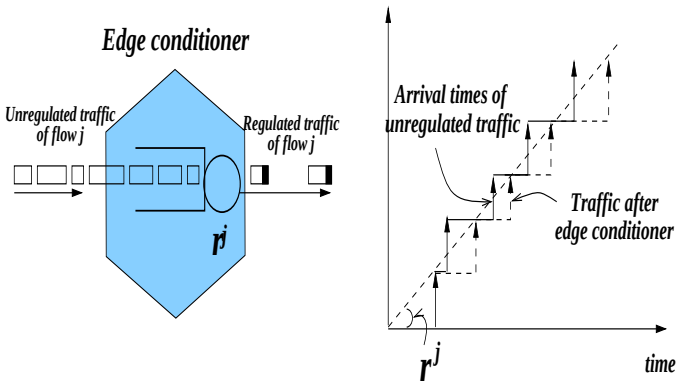
The packet state carried by a packet contains three types of information: 1) QoS reservation information of the flow[1] the packet belongs to (e.g., the reserved rate or delay parameter of the flow); 2) a virtual time stamp of the packet; and 3) a virtual time adjustment term. The packet state is initialized and inserted into a packet at the network edge after it has gone through the traffic conditioner. The *per-hop* behavior of each core router is defined with respect to the packet state carried by packets traversing it. As we will see later, *the virtual time stamps associated with the packets of a flow form the "thread" which "weaves" together the per-hop behaviors of core routers along the flow's path to support the QoS guarantee of the flow.*

Edge traffic conditioning plays a key role in the virtual time reference system, as *it ensures that traffic of a flow will never be injected into the network core at a rate exceeding its reserved rate.* This traffic conditioning is done by using a traffic shaper (or more specifically, a rate spacer, see Figure 4.1(b)), which enforces appropriate spacing between the packets of a flow based on its reserved rate. This is illustrated in the diagram on the right hand side of Figure 4.1(b). Formally, for a flow j with a reserved rate r^j, the inter-arrival time of two

[1]Here a flow can be either an individual user flow, or an aggregate traffic flow of multiple user flows, defined in whatever appropriate fashion.



(a) A conceptual network model



(b) Edge conditioner and its effect

Figure 4.1: Edge conditioning in the virtual time reference system.

consecutive packets of the flow is such that

$$a^{j,k+1} - a^{j,k} \geq \frac{L^{j,k+1}}{r^j}, \tag{4.1}$$

where $a^{j,k}$ denotes the arrival time of the kth packet $p^{j,k}$ of flow j at the network core, and $L^{j,k}$ the size of packet $p^{j,k}$.

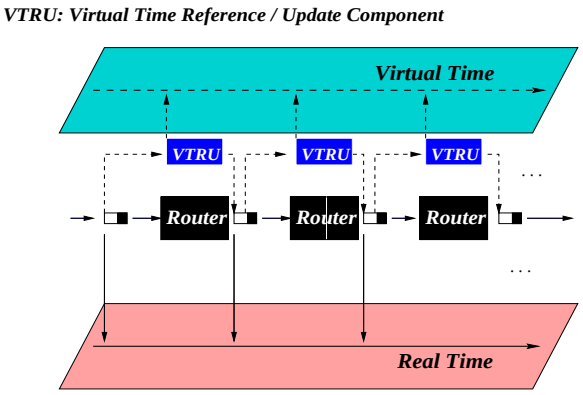
In the *conceptual* framework of the virtual time reference system, each core router is equipped with a per-hop virtual time reference/update mechanism to maintain the continual progression of the *virtual time* embodied by the packet virtual time stamps. As a packet traverses each core router along the path of its flow, a virtual time stamp is "attached" to the packet. This virtual time stamp represents the arrival time of the packet at the core router *in the virtual time*, and thus it is also referred to as the *virtual arrival time* of the packet at the core router. The virtual time stamps associated with packets of a flow satisfy an important property, which we refer to as the *virtual spacing property*. Let $\tilde{\omega}^{j,k}$ be the virtual time stamp associated with the kth packet, $p^{j,k}$, of flow j. Then

$$\tilde{\omega}^{j,k+1} - \tilde{\omega}^{j,k} \geq \frac{L^{j,k+1}}{r^j} \tag{4.2}$$

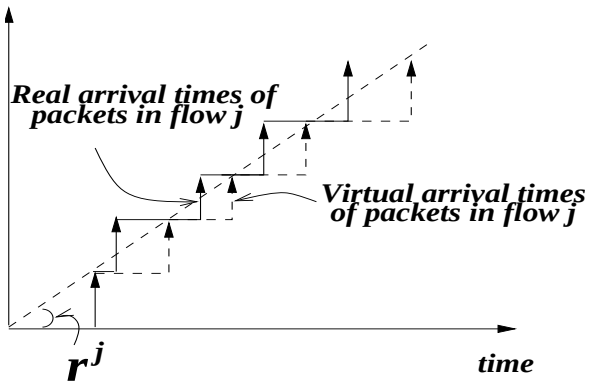
for all k.

Comparing (4.2) with (4.1), we see that *with respect to the virtual time*, the inter-arrival time spacing is preserved at a core router. Or to put it another way, *the "virtual rate" (as defined with respect to the virtual time) of packets of a flow arriving at a core router does not exceed the reserved rate of the flow*. Clearly this statement in general does not hold with respect to the *real* arrival times of the packets at a core router (see Figure 4.2(b)). Another key property of packet virtual time stamps is that *at a core router the virtual arrival time of a packet always lags behind its real arrival time*. This property (referred to as the *reality check condition*) is important in deriving end-to-end delay bound experienced by packets of a flow across the network core. The per-hop virtual time reference/update mechanism at a core router is designed in such a manner so as to ensure that these properties of the packet virtual time stamps are satisfied at the entry point and/or exit point of the core router (see the illustration in Figure 4.2).

The virtual time reference system provides a unifying framework to formalize the per-hop behavior of a core router and to quantify its ability to provide delay guarantees. This



(a) Virtual time reference/update mechanism



(b) Virtual traffic shaping

Figure 4.2: Illustration of the virtual time reference system.

formalism is independent of the scheduling mechanism employed by the core routers, *be it stateful or stateless*. Here we briefly describe how this mechanism works (see Section 4.4 for more details). Conceptually, for each packet traversing a core router, a *virtual finish time* is computed and assigned to it. This virtual finish time is derived from its virtual time stamp and other packet state information. Intuitively, it represents the time the packet finishes its service in an *ideal per-flow reference system*, where the flow to which the packet belongs to is the only flow serviced by the system. *The per-hop behavior of a core router is defined in terms of an upper bound on the difference between the actual departure time and virtual finish time of a packet traversing the core router*. This upper bound is referred to as the *error term* of the core router. Therefore, the scheduling mechanism of the core router can be abstracted into a *scheduling blackbox* characterized by an error term. This simple abstraction enables us to derive *end-to-end delay bounds* for flows traversing an arbitrary concatenation of such scheduling blackboxes, similar to what the notion of latency-rate servers [72] does for various fair queue algorithms.

In summary, while based on the DiffServ paradigm, the virtual time reference system renders the same expressive power and generality, in terms of the ability to provide guaranteed services, as the IntServ Model. Furthermore, the virtual time reference system provides a unifying scheduling framework upon which a scalable QoS provisioning and admission control framework can be built, where all QoS reservation states for guaranteed services are eliminated from the network core. The remainder of this chapter is devoted to laying formal foundation for the virtual time reference system. We also illustrate how various scheduling algorithms fit into the unifying framework. Issues of implementation and admission control will also be briefly discussed.

4.3 An Ideal Per-flow Virtual Time Reference System

In this section we motivate and introduce the notion of packet virtual time stamps in the context of an *ideal per-flow system*. The virtual time reference system defined in this context is then extended in the next section to account for the effect of packet scheduling in a real network system.

Figure 4.3 illustrates an ideal per-flow system, where a regulated flow is serviced by a dedicated channel. The dedicated channel consists of a series of servers in tandem. Packets of a flow j are serviced *in order* from server 1 to server h. For $k = 1, 2, \ldots$, the kth packet of flow j is denoted by $p^{j,k}$, and its size by $L^{j,k}$. Let r^j be the reserved rate of flow j, and d^j

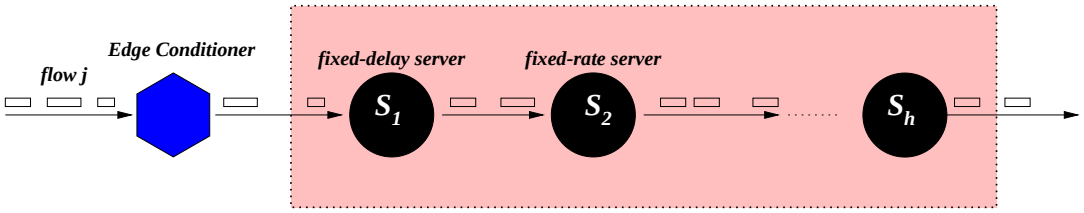


Figure 4.3: An ideal per-flow system.

a delay parameter associated with flow j. For simplicity of exposition, we assume that in this ideal per-flow system the propagation delay from one server to the next server is zero.

We consider two types of servers: *fixed-rate servers* and *fixed-delay servers*. A fixed-rate server has a service capacity equal to the reserved rate r^j of flow j. Hence a fixed-rate server takes $L^{j,k}/r^j$ amount of time to process packet $p^{j,k}$ of flow j. A fixed-delay server has a fixed latency, which equals to the delay parameter d^j of flow j. In other words, a fixed-delay server with latency d^j takes exactly d^j amount of time to process packets of flow j, independent of their packet sizes. We will see later the importance of fixed-delay servers in modeling scheduling algorithms that can provide delay-rate decoupling. Throughout this section, we assume that in the ideal per-flow system, there are q fixed-rate servers and $h-q$ fixed-delay servers.

Before we introduce the notion of packet virtual time stamps, we first need to understand and quantify the end-to-end delay experienced by the packets in the ideal per-flow system. We embark on this task in Section 4.3.1. Based on the results obtained thereof, in Section 4.3.2 we will introduce the ideal per-flow virtual time reference system. Table 4.1 summarizes the important notation used in the chapter.

4.3.1 End-to-end Delay of the Ideal Per-flow System

Recall that before entering this ideal per-flow system, packets from flow j go through an edge conditioner, where they are regulated so that the rate the packets are injected into the ideal per-flow system never exceeds the reserved rate r^j of the flow. Formally, let $a_1^{j,k}$ be the arrival time[2] of packet $p^{j,k}$ of flow j at the first server of the ideal per-flow system. Then

[2]Note that in order to model non-preemptive, non-cut-through network system, throughout the chapter we adopt the following convention: a packet is considered to have arrived at a server *only* when its last bit has been received, and it to have departed the server *only* when its last bit has been serviced.

Table 4.1: Notation used in VTRS.

General Notation	
$p^{j,k}$	the kth packet of flow j
$L^{j,k}$	packet length of $p^{j,k}$
$L^{j,max}$	maximum packet length of flow j
$L^{*,max}$	maximum packet length of all flows at a server/router
r^j	reserved rate of flow j
d^j	delay parameter of flow j
h	number of hops (servers/routers) along the path of flow j
q	number of fixed-rate servers/rate-based schedulers along flow j's path
Notation for the Ideal Per-Flow System	
$a_i^{j,k}$	arrival time of packet $p^{j,k}$ at node i
$f_i^{j,k}$	finish time of packet $p^{j,k}$ at node i
$\Delta_i^{j,k}$	cumulative queueing delay packet $p^{j,k}$ experienced up to server i (inclusive)
Notation for the Virtual Time Reference System	
$\tilde{\omega}_i^{j,k}$	virtual time stamp of packet $p^{j,k}$ at node i
$\tilde{\nu}_i^{j,k}$	virtual finish time of packet $p^{j,k}$ at node i
$\delta^{j,k}$	virtual time adjustment term for packet $p^{j,k}$: $\delta^{j,k} = \Delta_q^{j,k}/q$
$\tilde{d}_i^{j,k}$	virtual delay of packet $p^{j,k}$ at node i: $\tilde{d}_i^{j,k} = \tilde{\nu}_i^{j,k} - \tilde{\omega}_i^{j,k}$
$\hat{a}_i^{j,k}$	actual time packet $p^{j,k}$ arrives at node i
$\hat{f}_i^{j,k}$	actual time packet $p^{j,k}$ departs from node i
Ψ_i	error term of scheduling blackbox at node i
$\pi_{i,i+1}$	propagation delay from the i^{th} node to the $(i+1)^{th}$ node

the edge spacing condition (4.3) holds, namely,

$$a_1^{j,k+1} - a_1^{j,k} \geq L^{j,k+1}/r^j, \; k = 1, 2, \ldots \tag{4.3}$$

Let $A^j(\tau, t)$ denote the amount of flow j traffic that is injected into the ideal per-flow system over a time interval $[\tau, t]$. Using (4.3), it is easy to see that

$$A^j(\tau, t) \leq r^j(t - \tau) + L^{j,max} \tag{4.4}$$

where $L^{j,max}$ is the maximum packet size of flow j.

In order to derive the end-to-end delay experienced by packets in the ideal per-flow system, we first consider the *pure rate-based* system, where all servers are fixed-rate servers, i.e.,

$q = h$. This result can then be extended to the general ideal per-flow system with mixed fixed-rate and fixed-delay servers.

For $i = 1, 2, \ldots, h$, let $a_i^{j,k}$ denote the time packet $p^{j,k}$ arrives at server $\mathcal{S}_i$, and $f_i^{j,k}$ the time it leaves server i. In the pure rate-based ideal per-flow system, it is not hard to see that the following recursive relationships among $a_i^{j,k}$'s and $f_i^{j,k}$'s hold. For any $k = 1, 2, \ldots$,

$$a_i^{j,k} = f_{i-1}^{j,k},\ i = 2, \ldots, h, \tag{4.5}$$

and

$$f_i^{j,k} = \max\{a_i^{j,k}, f_i^{j,k-1}\} + \frac{L^{j,k}}{r^j},\ i = 1, 2, \ldots, h, \tag{4.6}$$

where in (4.6) we have used the convention that $f_i^{j,0} = 0$.

Note that in the special case where all packets of flow j have the same size L^j, each packet takes precisely L^j/r^j to be processed at each fixed-rate server. (In this case, a fixed-rate server functions as a fixed-delay server.) Because of the edge spacing property (4.3), we observe that no packet will ever be delayed in any fixed-rate server (see Figure 4.4(a)). In other words, for $i = 1, 2, \ldots, h$, $a_i^{j,k} \geq f_i^{j,k-1}$ and $f_i^{j,k} = a_i^{j,k} + L^j/r^j$. Therefore, in this case we have $f_h^{j,k} = a_1^{j,k} + hL^j/r^j$. Hence in this case, the *end-to-end delay* experienced by packet $p^{j,k}$ in the ideal per-flow system, which is defined as $f_h^{j,k} - a_1^{j,k}$, is hL^j/r^j.

In the general case where packets of flow j have variable sizes, the situation becomes somewhat more complicated. As shown in Figure 4.4(b), a small packet may be delayed at a server due to the longer processing time of a large packet preceding it. This delay can have a cascading effect which may cause more succeeding packets to be delayed.

For $i = 1, 2, \ldots, h$, let $\Delta_i^{j,k}$ denote the cumulative queueing delay experienced by packet $p^{j,k}$ up to server i (inclusive). Formally,

$$\Delta_i^{j,k} = f_i^{j,k} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j}). \tag{4.7}$$

For the pure rate-based ideal per-flow system, we can derive an important recursive relation, $\Delta_i^{j,k}$ to $\Delta_i^{j,k-1}$ and the arrival times of packets $p^{j,k-1}$ and $p^{j,k}$ at the first-hop server. This recursive relation is given in the following theorem, the proof of which can be found in Appendix B.1.

Theorem 7 *For any packet $p^{j,k}$, $k = 1, \ldots$, and $i = 1, 2, \ldots, h$,*

$$\Delta_i^{j,1} = 0$$

and

$$\Delta_i^{j,k} = \max\left\{0, \Delta_i^{j,k-1} + i\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}\right\}. \tag{4.8}$$

■

The importance of Theorem 7 lies in the fact that for each $p^{j,k}$, $\Delta_h^{j,k}$ can be calculated (recursively) *at the network edge*. As we will see in Section 4.3.2, this fact is critical in providing a *core stateless* definition of packet virtual time stamps for a system involving fixed-rate servers with variable packet sizes.

We now consider the general ideal per-flow system with both fixed-rate and fixed-delay servers. Recall that we assume we have q fixed-rate servers and $h - q$ fixed delay servers. As before, let $a_i^{j,k}$ and $f_i^{j,k}$ denote the arrival time and departure time of packet $p^{j,k}$ at server $\mathcal{S}_i$. Clearly, if $\mathcal{S}_i$ is a fixed-rate server, the recursive relation (4.6) holds among $a_i^{j,k}$'s and $f_i^{j,k}$'s. In the case where $\mathcal{S}_i$ is a fixed-delay server, we have that for $k = 1, 2, \ldots$,

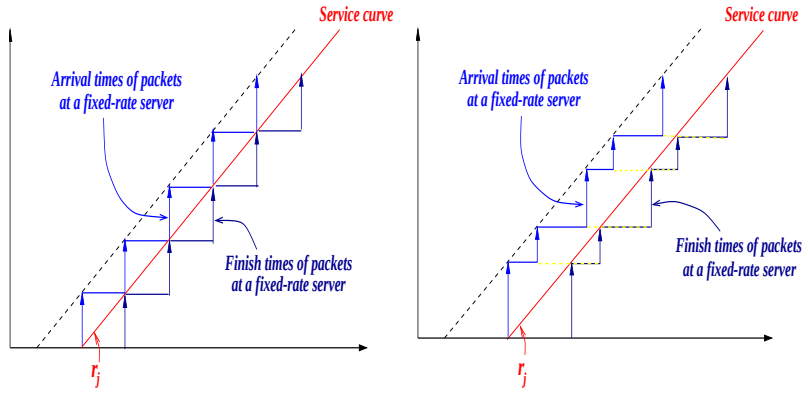
$$a_i^{j,k} = f_{i-1}^{j,k} \text{ and } f_i^{j,k} = a_i^{j,k} + d^j. \tag{4.9}$$

Unlike a fixed-rate server, every packet of flow j incurs a delay of precisely d^j at a fixed-delay server, regardless of its size. Hence there is no extra queueing delay due to the packet size difference (see Figure 4.4(c)). It is easy to see that we can re-arrange the location of the fixed-rate servers in the ideal per-flow system without affecting the end-to-end delay experienced by each packet in the system. Hence, without loss of generality, we can assume that the last $n - q$ servers are the fixed delay servers. Then from (4.7), we have

$$f_h^{j,k} = a_1^{j,k} + \Delta_q^{j,k} + q\frac{L^{j,k}}{r^j} + (h - q)d^j.$$

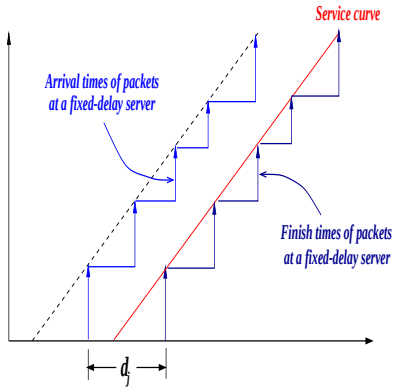
Therefore, the end-to-end delay of packet $p^{j,k}$ in the ideal per-flow system is $f_h^{j,k} - a_1^{j,k} = \Delta_q^{j,k} + q\frac{L^{j,k}}{r^j} + (h - q)d^j$. In particular, from Corollary 19 in Appendix B.1, we have $\Delta_q^{j,k} + qL^{j,k}/r^j \leq qL^{j,max}/r^j$. Thus,

$$f_h^{j,k} - a_1^{j,k} \leq q\frac{L^{j,max}}{r^j} + (h - q)d^j. \tag{4.10}$$



(a) Fixed-rate server with constant-size packets

(b) Fixed-rate server with variable-size packets



(c) Fixed-delay server

Figure 4.4: Delay experienced by packets at a server in the ideal per-flow system.

Note that the above end-to-end delay bound holds for all packets of flow j. As an aside, from the perspective of providing end-to-end delay bounds, we can treat a fixed-rate server with service capacity r^j as if it were a fixed-delay server with a latency parameter $L^{j,max}/r^j$. The resulting "pure" delay-based system yields exactly the same delay bound (4.10). This treatment of fixed-rate servers may simplify the implementation of the virtual time reference system in practice (see Section 4.8.1).

4.3.2 Packet Virtual Time Stamps and Ideal Per-flow System

The key construct in the proposed virtual time reference system is the notion of *packet virtual time stamps*. In this section, we formally specify the properties of packet virtual time

stamps, and provide a definition in the context of the ideal per-flow system. The resulting virtual time reference system is referred to as the *ideal per-flow virtual time reference system*.

For $i = 1, 2, \ldots, h$, let $\tilde{\omega}_i^{j,k}$ denote the virtual time stamp associated with packet $p^{j,k}$ at server $\mathcal{S}_i$. Intuitively, we can regard $\tilde{\omega}_i^{j,k}$ as the (virtual) arrival time of packet $p^{j,k}$ at server $\mathcal{S}_i$ *according to the virtual time*. At server $\mathcal{S}_i$, packet $p^{j,k}$ is also assigned a *virtual finish time*, denoted by $\tilde{\nu}_i^{j,k}$, where $\tilde{\nu}_i^{j,k} \geq \tilde{\omega}_i^{j,k}$. The difference $\tilde{d}_i^{j,k} = \tilde{\nu}_i^{j,k} - \tilde{\omega}_i^{j,k}$ is referred to as the *virtual delay* associated with packet $p^{j,k}$ at server $\mathcal{S}_i$.

We postulate the following properties that packet virtual time stamps (and the corresponding virtual finish times) of flow j must satisfy at each server $\mathcal{S}_i$.

Virtual Spacing: for $k = 1, 2, \ldots$,

$$\tilde{\omega}_i^{j,k+1} - \tilde{\omega}_i^{j,k} \geq \frac{L^{j,k+1}}{r^j}. \tag{4.11}$$

Reality Check: $\tilde{\omega}_i^{j,k} \geq a_i^{j,k}$, where recall that $a_i^{j,k}$ is the *real* time packet $p^{j,k}$ arrives at server $\mathcal{S}_i$.

Bounded Delay: $f_h^{j,k} = \tilde{\nu}_h^{j,k}$, or more generally, $f_h^{j,k} - \tilde{\nu}_h^{j,k}$ is bounded from above.

Core Stateless: the virtual time stamp $\tilde{\omega}_i^{j,k}$ of each packet $p^{j,k}$ can be calculated at each server $\mathcal{S}_i$ using solely the packet state information carried by the packet (possibly with some additional constant parameters associated with the server).

Intuitively, the *virtual spacing property* ensures that according to the virtual time, the amount of flow j traffic arriving at server i is limited by its reserved rate r^j. To put it formally, consider an arbitrary time interval $[\tau, t]$. We say that *according to the virtual time*, packet $p^{j,k}$ *arrives* at server $\mathcal{S}_i$ during the time interval $[\tau, t]$ (or simply, packet $p^{j,k}$ *virtually arrives* at server $\mathcal{S}_i$ during the time interval $[\tau, t]$), if and only if $\tau \leq \tilde{\omega}_i^{j,k} \leq t$. Let $\tilde{A}^j(\tau, t)$ denote the amount of flow j traffic arriving virtually in the time interval $[\tau, t]$. It can be shown that (see the *Virtual Shaping Lemma* and its proof in Appendix B.2)

$$\tilde{A}^j(\tau, t) \leq r^j(t - \tau) + L^{j,max}. \tag{4.12}$$

This bound is analogous to the traffic envelope (4.4) at the network edge, only that here the amount of flow j traffic is measured according to the virtual time. It suggests that if packet

virtual time stamps are used to schedule packets, *explicit rate control or reshaping within the network core is not necessary*.

The *reality check condition* and *bounded delay property* are important in ensuring that end-to-end delay bounds can be derived using the virtual time reference system (both for the ideal per-flow system as well as for a *real* network packet scheduling system, as we will see later). The *core stateless property* is the *key* to the construction of a *scalable* virtual time reference system that does not require per-flow scheduling state information at each core router.[3] In the following we provide a definition of packet virtual time stamps for the ideal per-flow system, and show that it satisfies all the four properties listed above.

Consider the ideal per-flow system shown in Figure 4.3. Recall that we assume that there are q fixed-rate servers and $h-q$ fixed-delay servers in the ideal per-flow system. For each packet $p^{j,k}$, define $\delta^{j,k} = \Delta_q^{j,k}/q$. We refer to $\delta^{j,k}$ as the *virtual time adjustment term* for packet $p^{j,k}$. It is calculated at the network edge and inserted into the packet state in addition to the reserved rate r^j and delay parameter d^j. For $i = 1, 2, \ldots, h$, the *virtual delay* $\tilde{d}_i^{j,k}$ associated with packet $p^{j,k}$ at server $\mathcal{S}_i$ is computed from the packet state information using the following formula:

$$\tilde{d}_i^{j,k} = \begin{cases} L^{j,k}/r^j + \delta^{j,k} & \text{if } \mathcal{S}_i \text{ is a fixed rate server,} \\ d^j & \text{if } \mathcal{S}_i \text{ a fixed delay server.} \end{cases} \tag{4.13}$$

At the first-hop server $\mathcal{S}_1$, the virtual time stamp of packet $p^{j,k}$ is defined to be $\tilde{\omega}_1^{j,k} = a_1^{j,k}$, which is the time packet $p^{j,k}$ is injected to the ideal per-flow system and arrives at $\mathcal{S}_1$. This value is inserted into the packet state of $p^{j,k}$ at the network edge. The corresponding virtual finish time of $p^{j,k}$ at server $\mathcal{S}_i$ is given by $\tilde{\nu}_i^{j,k} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k}$.

For $i = 2, \ldots, h$, the virtual time stamp $\tilde{\omega}_i^{j,k}$ and the corresponding virtual finish time $\tilde{\nu}_i^{j,k}$ associated with packet $p^{j,k}$ at server $\mathcal{S}_i$ are defined as follows:

$$\tilde{\omega}_i^{j,k} = \tilde{\nu}_{i-1}^{j,k} = \tilde{\omega}_{i-1}^{j,k} + \tilde{d}_{i-1}^{j,k} \text{ and } \tilde{\nu}_i^{j,k} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k}. \tag{4.14}$$

[3] As an example of the importance of the core stateless property, consider the following "definition" of virtual time stamps. At each server $\mathcal{S}_i$, we define $\tilde{\omega}_i^{j,k} = a_i^{j,k}$ and $\tilde{\nu}_i^{j,k} = f_i^{j,k}$ for packet $p^{j,k}$. Then from (4.3), (4.5), (4.6), and (4.9), we see that the virtual spacing property holds. Furthermore, the reality check condition and the bounded delay property also hold trivially. However, using this definition, computation of packet virtual time stamps at a core router requires maintenance of per-flow scheduling state (i.e., the value of the virtual time stamp of the previous packet). Therefore, such a definition is *not* core stateless.

From the above definition, it is clear that the core stateless property holds trivially. In the rest of this section we show that the other three properties are also satisfied. This fact is stated in the following theorem.

Theorem 8 *For $i = 1, 2, \ldots, h$, the virtual spacing property (4.11) holds at each server $\mathcal{S}_i$. Furthermore,*

$$\tilde{\omega}_i^{j,k} \geq a_i^{j,k} \tag{4.15}$$

and in particular,

$$\tilde{\nu}_h^{j,k} = f_h^{j,k}. \tag{4.16}$$

Proof: We first establish that the virtual spacing property holds. Fix i and let q_i be the number of fixed-rate servers along the path from the first hop to the $(i-1)$th hop. Clearly $q_i \leq q$. Note that from (4.14) and (4.13), we have

$$\tilde{\omega}_i^{j,k} = \tilde{\omega}_1^{j,k} + \sum_{q=1}^{i-1} \tilde{d}_q^{j,k} = a_1^{j,k} + q_i(\delta^{j,k} + \frac{L^{j,k}}{r^j}) + (i - 1 - q_i)d^j. \tag{4.17}$$

Hence to prove (4.11), it suffices to show

$$a_1^{j,k} + q_i(\delta^{j,k} + \frac{L^{j,k}}{r^j}) \geq a_1^{j,k-1} + q_i(\delta^{j,k-1} + \frac{L^{j,k-1}}{r^j}) + \frac{L^{j,k}}{r^j},$$

or equivalently,

$$\delta^{j,k} \geq \delta^{j,k-1} + \frac{L^{j,k-1} - L^{j,k}}{r^j} + \frac{a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}}{q_i}. \tag{4.18}$$

From the definition of $\delta^{j,k}$ and Theorem 7, we have

$$\delta^{j,k} = \frac{\Delta_q^{j,k}}{q} \geq \frac{\Delta_q^{j,k-1}}{q} + \frac{L^{j,k-1} - L^{j,k}}{r^j} + \frac{a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}}{q}. \tag{4.19}$$

Using (4.3) and the fact that $q_i \leq q$, we see that the last term in the right hand side of (4.19) is larger than the corresponding term in (4.18). Hence (4.18) holds.

We now establish (4.15). As $\tilde{\omega}_1^{j,k} = a_1^{j,k}$, (4.15) holds trivially for $i = 1$. To show that (4.15) also hold for $i = 2, \ldots, h$, observe that

$$a_i^{j,k} = f_{i-1}^{j,k} = a_1^{j,k} + \Delta_{q_i}^{j,k} + q_i \frac{L^{j,k}}{r^j} + (i - 1 - q_i)d^j, \tag{4.20}$$

where recall that q_i is the number of fixed-rate servers among $\mathcal{S}_1, \ldots, \mathcal{S}_{i-1}$.

Comparing (4.20) and (4.17) and using the fact that $\Delta_q^{j,k}/q \geq \Delta_{q_i}^{j,k}/q_i$ (see Corollary 18 in Appendix B.1), we see that (4.15) indeed holds.

Lastly, (4.16) follows easily from the definitions, as

$$\tilde{\nu}_h^{j,k} = a_1^{j,k} + \Delta_q^{j,k} + q \frac{L^{j,k}}{r^j} + (h - q)d^j = f_h^{j,k}. \tag{4.21}$$

■

4.4 Virtual Time Reference System and Packet Scheduling

In this section we extend the virtual time reference system defined in the context of the ideal per-flow system to a network system where each core router is shared by multiple flows. The key notion we will introduce is the *error term* of a core router (or rather, of its scheduling mechanism), which accounts for the effect of packet scheduling in providing delay guarantees for a flow. Based on this notion of error term, we define a generic virtual time reference system, which provides a unifying scheduling framework to characterize the end-to-end behavior of core routers in providing delay guarantees.

Consider a flow j, whose path through a network core is shown in Figure 4.5. Flow j has a reserved rate r^j and a delay parameter d^j. The traffic of flow j is regulated at the network edge such that for $k = 1, 2, \ldots$,

$$\hat{a}_1^{j,k+1} - \hat{a}_1^{j,k} \geq \frac{L^{j,k+1}}{r^j} \tag{4.22}$$

where $\hat{a}_1^{j,k}$ is the *actual* time packet $p^{j,k}$ of flow j arrives at the first router along its path, after being injected into the network core.

As shown in Figure 4.5, the path of flow j consists of h core routers, each of which employs certain scheduling mechanism to provide guaranteed service for flow j. For $i = 1, 2, \ldots, h$,

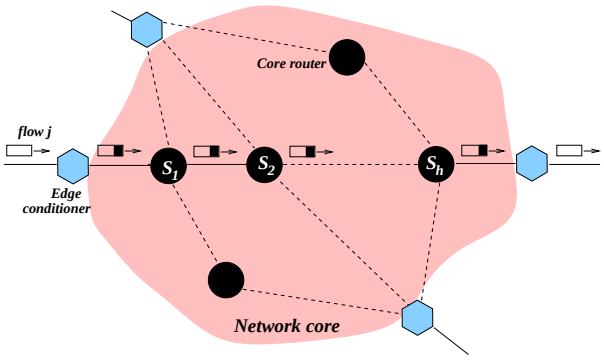


Figure 4.5: A flow traverses a network core.

we will refer to the scheduler at core router i as a scheduling *blackbox*, and denote it by $\mathcal{S}_i$. In the following, we will first characterize the *per-hop behavior* of the scheduling blackboxes, and then show how end-to-end delay bounds can be derived based on this characterization of their per-hop behavior.

4.4.1 Scheduling Blackbox: Per-Hop Behavior

Corresponding to the fixed-rate servers and fixed-delay servers in the ideal per-flow system, we categorize the scheduling blackboxes into two types: *rate-based* scheduling blackbox and *delay-based* scheduling blackbox. They are distinguished by how the virtual delay parameter is computed, as in the ideal per-flow system. For a rate-based scheduling blackbox $\mathcal{S}_i$, packet $p^{j,k}$ of flow j is assigned a virtual delay $\tilde{d}_i^{j,k} = L^{j,k}/r^j + \delta^{j,k}$, where $\delta^{j,k}$ is the virtual time adjustment term carried in the packet state. For a delay-based scheduling blackbox $\mathcal{S}_i$, packet $p^{j,k}$ of flow j is assigned a virtual delay $\tilde{d}_i^{j,k} = d^j$. In other words, the virtual delay $\tilde{d}_i^{j,k}$ is given by the same formula as in (4.13). In either case, we see that the virtual delay $\tilde{d}_i^{j,k}$ can be computed using only the packet state information carried by the packet.

Now fix an index i, where $i = 1, 2, \ldots, h$, and consider the scheduling blackbox $\mathcal{S}_i$. For any flow j traversing the scheduling blackbox $\mathcal{S}_i$, let $\tilde{\omega}_i^{j,k}$ be the virtual time stamp associated with packet $p^{j,k}$ as it enters $\mathcal{S}_i$. We will provide a definition for $\tilde{\omega}_i^{j,k}$ shortly and establish its properties. At this point, we only assume that the *reality check condition* holds at $\mathcal{S}_i$, namely,

$$\hat{a}_i^{j,k} \leq \tilde{\omega}_i^{j,k} \tag{4.23}$$

where $\hat{a}_i^{j,k}$ is the *actual* time that packet $p^{j,k}$ enters the scheduling blackbox $\mathcal{S}_i$. Hence upon

its arrival at $\mathcal{S}_i$, the virtual time stamp associated with packet $p^{j,k}$ is never smaller than its real arrival time.

At $\mathcal{S}_i$, packet $p^{j,k}$ is assigned a virtual finish time $\tilde{\nu}_i^{j,k}$, where $\tilde{\nu}_i^{j,k} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k}$. Let $\hat{f}_i^{j,k}$ denote the *actual* time packet $p^{j,k}$ departs $\mathcal{S}_i^{j,k}$, i.e., $\hat{f}_i^{j,k}$ is the *real finish time* of $p^{j,k}$. We say that the scheduling blackbox $\mathcal{S}_i$ can *guarantee* packets of flow j their virtual delays with an *error term* Ψ_i, if for any k,

$$\hat{f}_i^{j,k} \leq \tilde{\nu}_i^{j,k} + \Psi_i. \tag{4.24}$$

In other words, each packet is guaranteed to depart the scheduling blackbox $\mathcal{S}_i$ by the time $\tilde{\nu}_i^{j,k} + \Psi_i = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k} + \Psi_i$.

By using the packet virtual finish time as a reference point to quantify the real finish time of a packet at a core router, we are able to abstract and characterize the per-hop behavior of a core router via an error term. This error term captures the ability of the core router to provide guaranteed services to a flow. In particular, for a rate-based scheduling blackbox $\mathcal{S}_i$, we say that $\mathcal{S}_i$ guarantees flow j its reserved rate r^j with an error term Ψ_i if (4.24) holds. For a delay-based scheduling blackbox $\mathcal{S}_i$, we say that $\mathcal{S}_i$ guarantees flow j its delay parameter d^j with an error term Ψ_i if (4.24) holds.

4.4.2 Virtual Time Reference System and End-to-End Delay Bounds

We now extend the virtual time reference system defined earlier to account for the effect of packet scheduling by incorporating the error terms of core routers into the system. In particular, we illustrate how packet virtual time stamps associated with flow j should be referenced and updated as packets of flow j traverse the core routers along the flow's path. We also derive and characterize the end-to-end behavior of these core routers in concatenation.

Consider the path of flow j shown in Figure 4.5. Suppose there are q rate-based scheduling blackboxes and $h-q$ delay-based scheduling blackboxes. For $i = 1, 2, \ldots, h$, let Ψ_i be the error term associated with the scheduling blackbox $\mathcal{S}_i$. In other words, $\mathcal{S}_i$ can guarantee flow j either its reserved rate r^j or its delay parameter d^j with an error term Ψ_i. The virtual delay $\tilde{d}_i^{j,k}$ associated with packet $p^{j,k}$ at $\mathcal{S}_i$ is given below:

$$\tilde{d}_i^{j,k} = \begin{cases} L^{j,k}/r^j + \delta^{j,k} & \text{if } \mathcal{S}_i \text{ is rate-based,} \\ d^j & \text{if } \mathcal{S}_i \text{ is delay-based} \end{cases}$$

where $\delta^{j,k} = \Delta_q^{j,k}/q$ is the virtual time adjustment term of packet $p^{j,k}$.

For $i = 1, 2, \ldots, h$, let $\tilde{\omega}_i^{j,k}$ denote the virtual time stamp associated with packet $p^{j,k}$ of flow j at $\mathcal{S}_i$, and $\tilde{\nu}^{j,k}$ be the virtual finish time of packet $p^{j,k}$ at $\mathcal{S}_i$. Then

$$\tilde{\nu}^{j,k} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k}.$$

We now define $\tilde{\omega}_i^{j,k}$ and show that this definition satisfies the four requirements of packet virtual time stamps, namely, the *virtual spacing property*, the *reality check condition*, the *bounded delay property* and the *core stateless property*. Here in defining the reality check condition and bounded delay property, the quantities $a_i^{j,k}$ and $f_i^{j,k}$ defined in Section 4.3.2 are replaced by $\hat{a}_i^{j,k}$ and $\hat{f}_i^{j,k}$, which denote the *real arrival time* and *real finish time* of packet $p^{j,k}$ at $\mathcal{S}_i$, respectively.

As in the ideal per-flow system, the virtual time stamp associated with packet $p^{j,k}$ at the first-hop router $\mathcal{S}_1$ is set to its (real) arrival time, i.e.,

$$\tilde{\omega}_1^{j,k} = \hat{a}_1^{j,k}. \tag{4.25}$$

Thus $\tilde{\nu}_1^{j,k} = \tilde{\omega}_1^{j,k} + \tilde{d}_1^{j,k} = \hat{a}_1^{j,k} + \tilde{d}_1^{j,k}$.

From (4.22), the virtual spacing property is clearly met at the first-hop router. Furthermore, the reality check condition also holds trivially. Therefore, by the definition of Ψ_1, we have

$$\hat{f}_1^{j,k} \leq \tilde{\nu}_1^{j,k} + \Psi_1.$$

For $i = 1, 2, \ldots, h-1$, let $\pi_{i,i+1}$ denote the *propagation delay*[4] from the i^{th} hop router $\mathcal{S}_i$ to the $(i+1)^{th}$ hop router $\mathcal{S}_{i+1}$. Then

$$\hat{a}_{i+1}^{j,k} = \hat{f}_i^{j,k} + \pi_{i,i+1}.$$

By the definition of Ψ_i, we have

$$\hat{a}_{i+1}^{j,k} \leq \tilde{\nu}_i^{j,k} + \Psi_i + \pi_{i,i+1}. \tag{4.26}$$

[4]Here for simplicity, we assume that the propagation delay experienced by each packet of flow j from $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$ is a constant. In case this is not true, we can assume $\pi_{i,i+1}$ to be the *maximum* propagation delay from $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$. Then for any packet $p^{j,k}$, $\hat{a}_{i+1}^{j,k} \leq \hat{f}_i^{j,k} + \pi_{i,i+1}$.

In order to ensure that the reality check condition holds as packet $p^{j,k}$ enters the $(i+1)^{th}$ hop router $\mathcal{S}_{i+1}$, the relation (4.26) suggests that the virtual time stamp $\tilde{\omega}_{i+1}^{j,k}$ associated with packet $p^{j,k}$ at $\mathcal{S}_{i+1}$ should be defined as follows:

$$\tilde{\omega}_{i+1}^{j,k} = \tilde{\nu}_i^{j,k} + \Psi_i + \pi_{i,i+1} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k} + \Psi_i + \pi_{i,i+1}. \tag{4.27}$$

Then $\hat{a}_{i+1}^{j,k} \leq \tilde{\omega}_{i+1}^{j,k}$.

Since Ψ_i's and $\pi_{i,i+1}$'s are *fixed* parameters associated with the core routers and the path of flow j, it is clear that the packet virtual time stamps defined using (4.27) are *core stateless*. Namely, they can be computed at each core router using only the packet state information carried by the packets (in addition to the two fixed parameters associated with the routers and the flow's path). Thus *no per-flow state needs to be maintained at these core routers.*

Since $\Psi_i + \pi_{i,i+1}$ is a constant independent of $p^{j,k}$, comparing the definition of $\tilde{\omega}_i^{j,k}$ in (4.27) and that in (4.14), it is easy to see that the virtual spacing property also holds at each core router $\mathcal{S}_i$. Furthermore, we have

$$\tilde{\omega}_{i+1}^{j,k} = \tilde{\nu}_i^{j,k} + \Psi_i + \pi_{i,i+1} = \hat{a}_1^{j,k} + \sum_{q=1}^{i} \tilde{d}_q^{j,k} + \sum_{q=1}^{i} \Psi_q + \sum_{q=1}^{i} \pi_{q,q+1}.$$

In particular, we see that the bounded delay property holds, as

$$\hat{f}_h^{j,k} \leq \tilde{\nu}_h^{j,k} + \Psi_h = \hat{a}_1^{j,k} + \sum_{q=1}^{h} \tilde{d}_q^{j,k} + \sum_{q=1}^{h} \Psi_q + \sum_{q=2}^{h} \pi_{q-1,q}.$$

This completes the construction of packet virtual time stamps for flow j. In a nutshell, packet virtual time stamps are initialized using (4.25) at the network edge, and are referenced and updated using (4.27) at each core router. The reference and update mechanism of the resulting virtual time reference system is schematically shown in Figure 4.6.

Using the virtual time reference system, the following end-to-end delay bound for flow j can be easily derived from the bounded delay property of packet virtual time stamps:

$$\begin{aligned} \hat{f}_h^{j,k} - \hat{a}_1^{j,k} &\leq \Delta_q^{j,k} + q\frac{L^{j,k}}{r^j} + (h-q)d^j + \sum_{i=1}^{h} \Psi_i + \sum_{i=1}^{h-1} \pi_{i,i+1} \\ &\leq q\frac{L^{j,max}}{r^j} + (h-q)d^j + \sum_{i=1}^{h} \Psi_i + \sum_{i=1}^{h-1} \pi_{i,i+1}, \end{aligned} \tag{4.28}$$

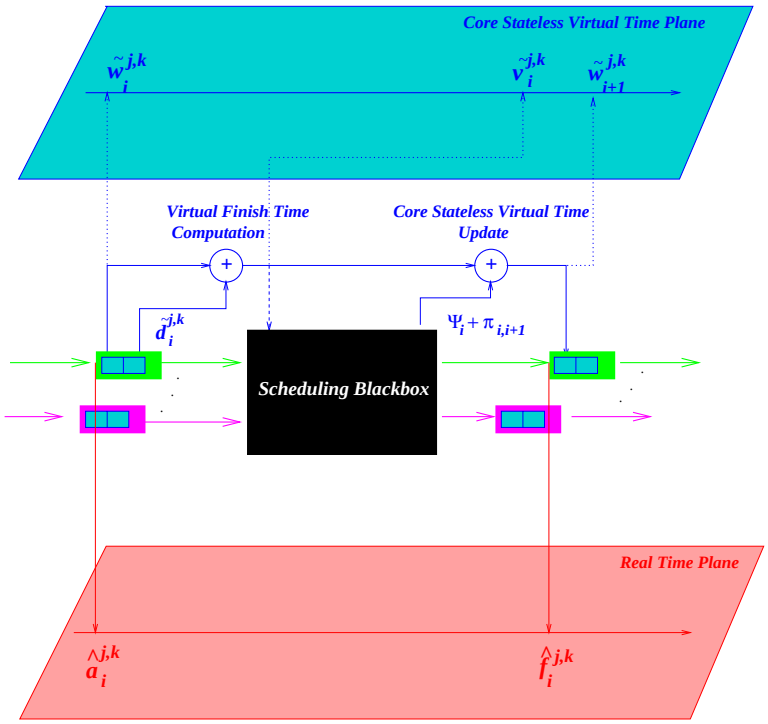


Figure 4.6: Virtual time reference system: per-hop behavior and operations.

where the last inequality follows from Corollary 19 in Appendix B.1.

In the special case where only rate-based scheduling algorithms are employed at core routers (i.e., $q = h$), we have

$$\hat{f}_h^{j,k} - \hat{a}_1^{j,k} \leq h\frac{L^{j,max}}{r^j} + \sum_{i=1}^{h}\Psi_i + \sum_{i=1}^{h-1}\pi_{i,i+1}. \tag{4.29}$$

This bound is analogous to those derived for fair-queueing/latency-rate-server based scheduling algorithms [30, 60, 72]. In particular, if $\Psi_i = L^{*,max}/C_i$, where $L^{*,max}$ is the maximum packet size permissible at the i^{th} router and C_i is its service capacity, then the above inequality yields precisely the same delay bound as is obtained for a flow in a network of Weighted Fair Queueing (WFQ) schedulers [60] (or Virtual Clock (VC) schedulers [30] for that matter).

By incorporating delay-based scheduling algorithms into our framework, we can provide a certain degree of *rate and delay decoupling*. To see this, let D^j be such that $D^j \geq$

$pL^{j,max}/r^j$. Set

$$d^j = \frac{1}{h-p}[D^j - p\frac{L^{j,max}}{r^j}]. \tag{4.30}$$

Suppose we can design delay-based scheduling algorithms that can support the delay parameter d^j for flow j. Then from (4.28) we have

$$\hat{f}_h^{j,k} - \hat{a}_1^{j,k} \leq D^j + \sum_{i=1}^{h} \Psi_i + \sum_{i=1}^{h-1} \pi_{i,i+1}.$$

In the special case where $q = 0$ (i.e., only delay-based scheduling algorithms are employed along the path of flow j), setting $d^j = D^j/h$ yields

$$\hat{f}_h^{j,k} - \hat{a}_1^{j,k} \leq D^j + \sum_{i=1}^{h} \Psi_i + \sum_{i=1}^{h-1} \pi_{i,i+1}.$$

Clearly, this delay bound is completely decoupled from the reserved rate of flow j. Hence using pure delay-based scheduling algorithms, it is possible to support an arbitrary delay bound $D^j \geq 0$ for flow j (apart from the constant term $\sum_{i=1}^{h} \Psi_i + \sum_{i=1}^{h-1} \pi_{i,i+1}$ associated with the path of flow j).

Before we leave this section, it is interesting to compare our virtual time reference system with the WFQ-based reference system used in the Internet IntServ model [6]. From (4.28), we see that the constant term $\sum_{i=1}^{h} \Psi_i + \sum_{i=1}^{h-1} \pi_{i,i+1}$ is equivalent to the D_{total} term defined in the IntServ guaranteed service, whereas the rate-dependent term $q\frac{L^{j,max}}{r^j}$ corresponds to the C_{total} term in the IntServ guaranteed service. Furthermore, the delay parameter d^j, similar to the *slack term* in the IntServ guaranteed service, can be used to take advantage of the delay-rate decoupling offered by delay-based scheduling algorithms. Therefore, in terms of providing delay guaranteed services, our virtual time reference system has essentially the same expressive power as the IntServ guaranteed service model. What distinguishes the virtual time reference system from the IntServ guaranteed service model is its *core stateless* nature. Using the Internet DiffServ paradigm and the notion of packet virtual time stamps, our virtual time reference system allows for more scalable scheduling mechanisms (e.g., *core stateless* scheduling algorithms) to be employed for the support of guaranteed services. In addition, our virtual time reference system can accommodate both core stateless and stateful scheduling algorithms, thereby making it a unifying scheduling framework.

In the next section we demonstrate that the notion of packet virtual time stamps leads to the design of new *core stateless* scheduling algorithms. In particular, we design both rate-based and delay-based scheduling algorithms with the *minimum* error term $\Psi = L^{*,max}/C$. In Section 4.6 we illustrate how some simple static scheduling algorithms can be used in our framework to provide scalable scheduling support with resource pre-configuration. In Section 4.7 we show that the generic scheduling framework based on latency-rate servers (examples of which include VC, WFQ and various variations of fair queueing algorithms) can also be accommodated in our framework.

4.5 Core Stateless Scheduling Algorithms: Examples

In this section we illustrate how the notion of packet virtual time stamps can be used to design new *core stateless* scheduling algorithms. In particular, we design a number of rate-based and delay-based core stateless scheduling algorithms, and establish their error terms using the properties of packet virtual time stamps.

4.5.1 Rate-Based Core Stateless Scheduling Algorithms

4.5.1.1 Core Stateless Virtual Clock Scheduling Algorithm

A *core stateless virtual clock* ($C_{\not S}$VC) scheduler $\mathcal{S}$ is a rate-based scheduler. It services packets in the order of their virtual finish time. For any packet $p^{j,k}$ traversing $\mathcal{S}$, let $\tilde{\omega}^{j,k}$ be the virtual time carried by $p^{j,k}$ as it enters $\mathcal{S}$, and $\tilde{d}^{j,k} = \frac{L^{j,k}}{r^j} + \delta^{j,k}$ be its virtual delay. Then the virtual finish time $\tilde{\nu}^{j,k}$ of $p^{j,k}$ is given by $\tilde{\omega}^{j,k} + \tilde{d}^{j,k}$. We claim that the $C_{\not S}$VC scheduler can guarantee each flow j its reserved rate r^j with the minimum error term $\Psi_{C_{\not S}VC} = L^{*,max}/C$, provided that an appropriate schedulability condition is met. This fact is stated formally in the following theorem, the proof of which can be found in Appendix B.2.

Theorem 9 *Consider N flows traversing a $C_{\not S}$VC scheduler $\mathcal{S}$ such that the schedulability condition $\sum_{j=1}^{N} r^j \leq C$ is satisfied. Suppose that $\hat{a}^{j,k} \leq \tilde{\omega}^{j,k}$ for any packet $p^{j,k}$ of flow j, $j = 1, 2, \ldots, N$. Then*

$$\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \frac{L^{*,max}}{C}. \tag{4.31}$$

In other words, $\Psi_{C_{\not S}VC} = \frac{L^{*,max}}{C}$.

4.5.1.2 Core-Jitter Virtual Clock Scheduling Algorithm

In [73], the core-jitter virtual clock (CJVC) scheduling algorithm is presented, which can be considered as a non-work-conserving version of the $C_{\$}$VC scheduling algorithm. In the CJVC scheduling algorithm, if a packet $p^{j,k}$ arrives too early, it is held in a rate controller until its *eligibility time*, namely, when the real time reaches $\tilde{e}^{j,k} = \tilde{\omega}^{j,k} + \delta^{j,k}$ (see Appendix B.2). It can be shown that the CJVC scheduler has the same error term as the $C_{\$}$VC scheduler does, i.e., $\Psi_{CJVC} = L^{*,max}/C$. The proof follows a similar argument as used in the case of $C_{\$}$VC. Comparing $C_{\$}$VC and CJVC, we see that the *work-conserving* $C_{\$}$VC scheduling algorithm provides the same delay bound as the *non-work-conserving* CJVC scheduling algorithm, without the additional complexity of rate controller (*albeit core stateless*) at each core router as is required by CJVC. This is achieved because of the virtual shaping property of the core stateless virtual time.

4.5.1.3 Approximation to Core Stateless Virtual Clock

In order to reduce the overhead of sorting, we can use a series of FIFO queues to approximate the $C_{\$}$VC scheduling algorithm. These FIFO queues are referred to as *calendar queues*, as they are labeled by discrete time epochs. Conceptually, the virtual time is divided into fixed time slots with a slot unit of ι: $\tau_0, \tau_1, \ldots, \tau_p, \ldots$, where $\tau_p = p\iota$. Each time slot τ_p has an associated FIFO queue[5]. Upon its arrival, a packet $p^{j,k}$ is placed in the queue associated with time slot τ_p if $\tau_p \leq \tilde{\nu}^{j,k} < \tau_{p+1}$. The calendar queues are serviced in the order of τ_p. Moreover, if queue τ_p is empty, the packet at the head of line from the next non-empty queue is serviced. However, suppose a new packet arrives at the previously empty queue τ_p while this packet (from a queue $\tau_{p'}$ where $\tau_{p'} > \tau_p$) is being serviced. After this packet departs, queue τ_p will be serviced next.

We call the above calendar queue approximation to $C_{\$}$VC the *slotted* $C_{\$}$VC. It can be shown that a slotted $C_{\$}$VC scheduler has an error term $\Psi_{slotted\text{-}C_{\$}VC} = L^{*,max}/C + \iota$. This result is stated in the following Theorem, the proof of which can be found in Appendix B.2.

Theorem 10 *Consider N flows traversing a slotted-$C_{\$}$VC scheduler $\mathcal{S}$ such that the schedulability condition $\sum_{j=1}^{N} r^j \leq C$ is satisfied. Suppose that $\hat{a}^{j,k} \leq \tilde{\omega}^{j,k}$ for any packet $p^{j,k}$ of*

[5]In reality, only a limited number of FIFO queues are needed. Whenever the current time t passes τ_p, the queues can be reused for the future time slots. This implementation is similar to the rotation priority queue proposed by Lieberherr *et al* [54, 55].

flow j, $j = 1, 2, \ldots, N$. Then

$$\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \frac{L^{*,max}}{C} + \iota. \tag{4.32}$$

In other words, $\Psi_{slotted\text{-}C\text{-}SVC} = \frac{L^{*,max}}{C} + \iota$.

4.5.2 Delay-Based Core Stateless Scheduling Algorithms

4.5.2.1 Virtual Time Earliest Deadline First Algorithm

A *virtual time earliest deadline first* (VT-EDF) scheduler $\mathcal{S}$ is a delay-based scheduler. It services packets in the order of their virtual finish time. Recall that the virtual finish time of $p^{j,k}$ is given by $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k} + d^j$, where $\tilde{\omega}^{j,k}$ is the virtual time carried by $p^{j,k}$ as it enters $\mathcal{S}$ and d^j is the delay parameter associated with its flow. Provided that an appropriate schedulability condition is met, it can be shown that the VT-EDF scheduler can guarantee each flow j its delay parameter d^j with the minimum error term $\Psi_{VT\text{-}EDF} = L^{*,max}/C$. This fact is stated formally in the following theorem, the proof of which is relegated to Appendix B.2.

Theorem 11 *Consider N flows traversing a VT-EDF scheduler $\mathcal{S}$, where d^j is the delay parameter associated with flow j, $1 \leq j \leq N$. Without loss of generality, assume $0 \leq d^1 \leq d^2 \leq \cdots \leq d^N$. Suppose the following* schedulability condition *holds:*

$$\sum_{j=1}^{N} [r^j(t - d^j) + L^{j,max}] \mathbf{1}_{\{t \geq d^j\}} \leq Ct, \text{ for any } t \geq 0 \tag{4.33}$$

where the indicator function $\mathbf{1}_{\{t \geq d^j\}} = 1$ if $t \geq d^j$, 0 otherwise. Suppose that $\hat{a}^{j,k} \leq \tilde{\omega}^{j,k}$ for any packet $p^{j,k}$ of flow j, $j = 1, 2, \ldots, N$. Then

$$\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \frac{L^{*,max}}{C}. \tag{4.34}$$

In other words, $\Psi_{VT\text{-}EDF} = \frac{L^{*,max}}{C}$.

We observe that the schedulability condition is essentially the same as the one derived for the standard EDF scheduling algorithm with flow traffic envelopes of the form given in (4.4) (see, e.g., [33, 55]). When using the standard EDF scheduling algorithm in a network to

support guaranteed delay services, *per-hop re-shaping at core routers is required*. This is to ensure that the flows conform to their traffic envelopes so that the schedulability condition still holds at each hop [34, 82]. This rendition of the standard EDF is sometimes referred to as rate-controlled EDF, or RC-EDF. In contrast, using VT-EDF only requires shaping at the network edge to control the rate of each flow (i.e., to ensure (4.3) holds). As long as the the schedulability condition (4.33) holds at every VT-EDF scheduler, *no per-hop re-shaping is needed for any core router*. This is because the VT-EDF scheduler services packets using the packet virtual arrival times, not their real arrival times. From (4.12), we see that *according to the virtual time*, the traffic envelope for each flow is still preserved at each core router.

It is also interesting to compare the schedulability condition of the Core Stateless Virtual Clock with that of the Virtual-Time EDF. For each flow j, set $d^j = L^{j,max}/r^j$. Then the condition $\sum_{j=1}^{N} r^j \leq C$ is equivalent to $\sum_{j=1}^{N}[r^j(t-d^j) + L^{j,max}] \leq Ct$ for all $t \geq 0$. Comparing this condition with (4.33), we see that for the *same* delay parameters $d^j = L^{j,max}/r^j$, the schedulability condition of Core Stateless Virtual Clock follows from that of VT-EDF.

In fact, because the left hand side of (4.33) is a piece-wise linear function, the schedulability condition (4.33) can be simplified into the following set of *closed-form* conditions on the rates r^j's and delay parameters d^j's of the flows:

$$\sum_{j=1}^{N} r^j \leq C$$

and for $j = 1, 2, \ldots, N$,

$$d^j \geq \frac{L^{j,max} + \sum_{m=1}^{j-1}[L^{m,max} - r^m d^m]}{C - \sum_{m=1}^{j-1} r^m}.$$

In particular, if $d^j = L^{j,max}/r^j$, $j = 1, 2, \ldots, N$, then the above conditions on d^j's hold trivially if the condition on r^j's holds, i.e, $\sum_{j=1}^{N} r^j \leq C$. Hence for this set of delay parameters, the rate condition $\sum_{j=1}^{N} r^j \leq C$ is sufficient. This is why the core stateless virtual clock can guarantee each flow a maximum delay $d^j = L^{j,max}/r^j$ as long as $\sum_{j=1}^{N} r^j \leq C$. In general, the VT-EDF scheduler can guarantee each flow a delay parameter d^j as long as the above set of conditions on the rates r^j's and delay parameters d^j's are satisfied.

4.5.2.2 Calendar Queue Approximation to VT-EDF

As in the case of the core stateless virtual clock scheduling algorithm, we can also design a calendar-queue approximation to the VT-EDF scheduling algorithm. This approximation scheme is referred to as the *slotted* VT-EDF scheduling algorithm. This scheme works exactly the same as the slotted $C_{\mathcal{S}}$VC scheduling algorithm, except that the virtual finish time $\tilde{\nu}^{j,k}$ of a packet $p^{j,k}$ is computed using $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k} + d^j$ instead of $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k} + L^{j,k}/r^j + \delta^{j,k}$.

The schedulability condition for the slotted VT-EDF scheduling algorithm becomes

$$\sum_{j=1}^{N}[r^j(t+\iota-d^j)+L^{j,max}]\mathbf{1}_{\{t\geq d^j\}} \leq Ct, \text{ for any } t \geq 0.$$

We claim that if the above schedulability condition holds, then a slotted-VT-EDF scheduler can guarantee flow j its delay parameter with an error term $\Psi_{slotted\text{-}VT\text{-}EDF} = L^{*,max}/C+\iota$.

4.5.3 Virtual Time Rate Control and Delay-Based Schedulers

Any delay-based scheduling algorithm is *in essence* core stateless in that it does not need to maintain *per-flow scheduling state* for determining when to service a packet so long as the delay parameter can be inferred from the packet directly (say, either from an explicit delay parameter carried by the packet or implicit from the flow label carried by the packet). What makes *conventional* delay-based scheduling algorithms such as EDF *stateful* is the need to perform *per-flow shaping* at each hop in order to support guaranteed delay services. For example, in the rate-controlled EDF (RC-EDF) scheduling algorithm, the rate controller needs to maintain per-flow state information to ensure that the flow traffic envelope process is satisfied at each hop. The VT-EDF scheduler proposed in Section 4.5.2.1 circumvents this problem by scheduling packets using their virtual finish time, a quantity that can be computed directly from the packet state information. An alternative approach is to use a *(core stateless) virtual time rate controller*, which is described below. Replacing the conventional rate controller with this virtual time rate controller, we can convert any rate-controlled, delay-based scheduling algorithm into a *core stateless* scheduler.

The operation of a virtual time rate controller is very simple. Upon the arrival of packet $p^{j,k}$, the virtual time rate controller assigns to it an *eligibility time* $e^{j,k}$, which is equal to its virtual time stamp, i.e., $e^{j,k} = \tilde{\omega}^{j,k}$. The packet is held at the virtual time rate controller until the real time reaches its eligibility time. The packet is then released to the delay-based

scheduler. Clearly, this virtual time rate controller does not need to maintain any per-flow state information. It can be easily implemented using a sorted list and a single timer.

For any flow j, let $A^j_{RT-RC}(\tau, t)$ denote the amount of flow j traffic released by the virtual time rate controller into the scheduler over any time interval $[\tau, t]$. We claim that

$$W^j_{VT\text{-}RC}(\tau, t) \leq r^j(t - \tau) + L^{j,max}. \tag{4.35}$$

This is because any packet $p^{j,k}$ released by the virtual time rate controller to the scheduler during the time interval $[\tau, t]$ must satisfy the condition: $\tau \leq \tilde{\omega}^{j,k} \leq t$. Applying the Virtual Shaping Lemma in Appendix B.2 to the time interval $[\tau, t]$, we see that (4.35) holds.

As an example to illustrate its usefulness, we design a new delay-based core stateless scheduler by incorporating the virtual time rate controller into the conventional static priority scheduling algorithm. The resulting core stateless scheduling algorithm is referred to as the *virtual-time rate-controlled static priority* (VT-RC-SP) scheduler. Its "stateful" counterpart is the rate-controlled static priority (RC-SP) scheduler proposed in [82].

In the following we present the schedulability condition and the error term for the VT-RC-SP scheduler. For $1 \leq q \leq M$, let D_q be a delay parameter associated with queue q, where $0 \leq D_1 < D_2 < \cdots < D_M$. Suppose a packet from flow j with a delay parameter d^j is placed into queue q if $D_q \leq d^j < D_{q+1}$. Let $\mathcal{F}_q$ be the set of flow j sharing queue q. Assuming that the following condition on D_q's holds:

$$D_q \geq \frac{\sum_{p=1}^{q} \sum_{j \in \mathcal{F}_p} L^{j,max}}{C - \sum_{p=1}^{q-1} \sum_{j \in \mathcal{F}_p} r^j} \text{ for } q = 1, 2, \ldots, M,$$

then the VT-RC-SP scheduler can guarantee each flow j in $\mathcal{F}_q$ a delay parameter $D_q \leq d^j$ with an error term $\Psi_{VT\text{-}RC\text{-}SP} = L^{*,max}/C$.

The above schedulability condition for VT-RC-SP can be established by applying the schedulability result obtained for RC-SP in [55] with the traffic envelope (4.35).

Lastly, we comment that we can also combine the virtual time rate controller with a rate-based scheduler. The core jitter virtual clock (CJVC) scheduling algorithm proposed in [73] and discussed in Section 4.5.1.2 is such an example. Note that in this case, the virtual time rate controller has a somewhat *different* definition: the eligibility time $\tilde{e}^{j,k}$ of a packet is defined as $\tilde{\omega}^{j,k} + \delta^{j,k}$ instead of $\tilde{\omega}^{j,k}$(See the Virtual Rate Control Lemma in Appendix B.2).

4.6 Static Scheduling Algorithms with Resource Pre-configuration

Another way to achieve the objective of scalable scheduling is to employ *static* scheduling algorithms. Here by a *static* scheduler we mean a scheduling algorithm which does not *directly* use flow-dependent packet state information such as the packet virtual time stamp, reserved rate or delay parameter of a flow. Examples of static scheduling algorithms are FIFO or simple priority-based scheduling schemes. In contrast, scheduling algorithms which *do* use this information are referred as *dynamic*. For example, the core stateless scheduling algorithms designed in the previous section are dynamic. Observe that static scheduling algorithms by definition are *core stateless*, as no per-flow states need to be maintained. Static scheduling algorithms are typically employed to support *traffic aggregation* and to provide *class of services*. In the following, we will use a number of examples to illustrate how static scheduling algorithms can be accommodated in our framework. In these examples, we assume that resources associated with the scheduler are *pre-configured*, namely, they are not dynamically allocated or de-allocated to flows as they arrive or depart.

4.6.1 FIFO

An FIFO scheduler $\mathcal{S}$ services packets in the order of their actual arrival time, regardless of their virtual time. We can view an FIFO scheduler as a delay-based scheduler with a *fictitious* delay parameter $d^j = 0$ assigned to each flow passing through the scheduler. We now determine the error term Ψ_{FIFO} for an FIFO scheduler with pre-configured service and buffer capacities.

Consider an FIFO scheduler $\mathcal{S}$ with a service capacity C and a total buffer size B. Suppose that N flows share $\mathcal{S}$, where the sum of the reserved rates $\sum_{j=1}^{N} r^j \leq C$. Furthermore, we assume that the buffer capacity B is appropriately provisioned such that no packet from any flow will ever be lost. (We will illustrate how to provision buffer and bandwidth resources for scheduling algorithms such as FIFO under the virtual time reference system in future work.) For any packet $p^{j,k}$, let $\hat{a}^{j,k}$ be the actual arrival time at the scheduler, and $\hat{f}^{j,k}$ be its actual departure time. It is clear that $\hat{f}^{j,k} \leq \hat{a}^{j,k} + B/C + L^{*,max}/C$. Since $\tilde{\omega}^{j,k} \geq \hat{a}^{j,k}$ and $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k}$, we have

$$\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \frac{B}{C} + \frac{L^{*,max}}{C}.$$

Therefore $\Psi_{FIFO} = B/C + L^{*,max}/C$.

By updating the virtual time stamp of packet $p^{j,k}$ using $\tilde{\omega}^{j,k}_{next\text{-}hop} = \tilde{\omega}^{j,k} + \frac{B}{C} + \frac{L^{*,max}}{C} + \pi$, where π is the propagation delay to the next hop of packet $p^{j,k}$, it is clear that $\tilde{\omega}^{j,k}_{next\text{-}hop}$ not only preserves the virtual spacing property of packet virtual time stamps but also meets the reality check condition at the next hop.

We can also use the FIFO scheduling algorithm as a *dynamic* delay-based scheduler with a fixed delay parameter: only flows with a delay parameter d^j such that $d^j \geq B/C$ can be supported. Under this interpretation, $\Psi_{FIFO} = L^{*,max}/C$. Like other dynamic delay-based schedulers, the virtual time stamp of a packet is updated as follows: $\tilde{\omega}^{j,k}_{next\text{-}hop} = \tilde{\omega}^{j,k} + d^j + \frac{L^{*,max}}{C} + \pi$. Clearly, a scheduler needs to choose one of the two interpretations so that the packet virtual time stamps can be updated appropriately and consistently.

4.6.2 Static WFQ with Pre-Configured Rates

To provide more service differentiation than the simple FIFO scheduler, we can employ a fixed number of FIFO queues with pre-configured service and buffer capacities. Bandwidth sharing among the FIFO queues can be implemented using, e.g., a Weighted Fair Queuing (WFQ) scheduler, or any of its variations. Suppose we have M FIFO queues which are serviced by a WFQ scheduler with a total service capacity of C. For $q = 1, 2, \ldots, M$, let $C_q = \phi_q C$ be the pre-configured service rate of queue q, where $0 \leq \phi_q \leq 1$ and $\sum_{q=1}^{M} \phi_q = 1$. In other words, each queue q is guaranteed a minimum service rate of C_q. In addition, we assume that the buffer capacity for queue q is B_q.

Let $\mathcal{F}_q$ denote the class of flows sharing queue q. We assume the schedulability condition $\sum_{j \in \mathcal{F}_q} r^j \leq C_q$ holds for each queue q. Furthermore, the buffer capacity B_q is also appropriately provisioned such that no packet from any flow in $\mathcal{F}_q$ will ever be lost. Then using the same argument as in the FIFO case, we can show that each queue has an error term $\Psi_q = B_q/C_q + L^{*,max}/C$. Namely, for any packet $p^{j,k}$ of flow $j \in \mathcal{F}_q$,

$$\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \frac{B_q}{C_q} + \frac{L^{*,max}}{C}$$

where as in the case of FIFO, we define $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k}$.

For $1 \leq q \leq M$, let $D_q = B_q/C_q$. Without loss of generality, assume that $0 \leq D_1 < D_2 < \cdots < D_M$. Then $\Psi_q = D_q + L^{*,max}/C$. In other words, using this static WFQ scheduler with multiple FIFO queues, we can support a fixed number of guaranteed delay classes. Note that using this multiple-queue WFQ scheme, flows from different queues can have

different error terms. Hence the packet virtual time stamp must be updated accordingly, depending on from which queue it is serviced.

As in the case of FIFO, we can also view this multiple-queue WFQ scheme as a *dynamic* delay-based scheduler with a set of fixed delay parameters: a flow with a delay parameter d^j can be supported and placed into queue q, $1 \leq q \leq M$, if $D_{q-1} \leq d^j < D_q$. Under this interpretation, the multiple-queue scheme has an error term $\Psi_{FIFO} = L^{*,max}/C$. Like other dynamic delay-based schedulers, the virtual time stamp of a packet is updated as follows: $\tilde{\omega}^{j,k}_{next\text{-}hop} = \tilde{\omega}^{j,k} + d^j + \frac{L^{*,max}}{C} + \pi$.

4.6.3 Static Priority with Pre-Configured Rates

As an alternative to the static WFQ scheduler with multiple FIFO queues, we can also implement a static priority scheduler with multiple FIFO queues. Again suppose we have M FIFO queues, where queues of lower index have higher priority. Namely, queue 1 has the highest priority, whereas queue M has the lowest priority. The queues are serviced in the order of their priorities: whenever queue q is not empty, it is serviced before queue $q+1$, $q+2$, ..., q_M. The scheduler is assumed to be non-preemptive.

The queues of the static priority scheduler is configured and provisioned in the following manner. For $q = 1, 2, \ldots, M$, $C_q \geq 0$ is a *nominal* service rate assigned to queue q, where $\sum_{q=1}^{M} C_q = C$. The nominal service rate of a queue is used to control the aggregate rate of flows sharing the queue. Let $\mathcal{F}_q$ denote the class of flows sharing queue q, we control the number of flows sharing queue q to ensure that $\sum_{j \in \mathcal{F}_q} r^j \leq C_q$ holds for each queue q. Furthermore, the buffer capacity B_q is also appropriately provisioned such that no packet from any flow in $\mathcal{F}_q$ will ever be lost.

For $1 \leq q \leq M$, let D_q be a delay parameter associated with queue q. We assume that $0 \leq D_1 < D_2 < \cdots < D_M$. Using the result obtained in [55], we can show the following schedulability condition. If the following condition on $D'_q s$ holds:

$$D_q \geq \frac{\sum_{p=1}^{q} B_p}{C - \sum_{p=1}^{q-1} C_p} \text{ for } q = 1, 2, \ldots, M,$$

then each queue has an error term $\Psi_q = \frac{B_q}{C_q} + \frac{L^{*,max}}{C}$. Namely, for any packet $p^{j,k}$ of flow $j \in \mathcal{F}_q$,

$$\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \frac{B_q}{C_q} + \frac{L^{*,max}}{C},$$

where $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k}$.

Clearly, we can also view this static priority queue scheme as a dynamic scheduler with an error term $\Psi = L^{*,max}/C$. The decision for placing a flow into an appropriate queue and mechanism for updating packet virtual time stamps are exactly the same as used in the static WFQ scheme with multiple queues.

4.7 Latency-Rate Servers and the Virtual Time Reference System

The virtual time reference system proposed in this chapter does not exclude the use of *stateful* scheduling algorithms, namely, those scheduling algorithms that maintain per-flow state information in order to provide guaranteed services. Such per-flow state information, for example, is imperative if "bounded fairness" in bandwidth sharing among flows is desired, in addition to delay and rate guarantees. To accommodate these stateful scheduling algorithms into our framework, it suffices to identify the error term incurred by these stateful scheduling algorithms. As an example to show how this can be done generally, we consider the class of scheduling algorithms introduced in [70, 72]—the *latency-rate servers*. This class encompasses virtually all known fair-queueing algorithms and its variations.

In defining a latency-rate server, a key notion introduced in [72] is the concept of *burst period*[6]. For any flow j, a *flow j burst period* is a maximal time interval $(\tau_1, \tau_2]$ such that for any time $t \in (\tau_1, \tau_2]$, packets of flow j arrive with rate greater than or equal to its reserved rate r^j, or

$$A^j(\tau_1, t) \geq r^j(t - \tau_1) \tag{4.36}$$

where $A^j(\tau_1, t)$ denote the amount of flow j traffic arriving during the timer interval $[\tau_1, t]$.

Consider a server $\mathcal{S}$. Suppose that the mth burst period of flow j starts at time τ. Let τ^* be the time that the last packet of the mth burst period of flow j departs server $\mathcal{S}$. For $\tau \leq t \leq \tau^*$, denote by $W^{j,m}(\tau, t)$ the total service provided to the packets of the pth burst period of flow j up to time t by server $\mathcal{S}$. We say $\mathcal{S}$ is a latency-rate server (with respect to flow j) if and only if for any time t, $\tau \leq t \leq \tau^*$,

$$W^{j,m}(\tau, t) \geq \max\{0, r^j(t - \tau - \Theta^j)\},$$

[6] Actually, the term "busy period" instead of "burst period" is used in [70, 72]. In order to avoid confusion with the standard definition and usage of "busy period" in queueing theory, we opt to use the term "burst period." Incidentally, the term "backlogged period" is used in [70, 72] to refer to the standard notion of "busy period."

where Θ^j is the minimum non-negative number such that the above inequality holds. It is referred to as the *latency* of server $\mathcal{S}$.

To relate a latency-rate server to the virtual time reference system, we provide an alternative definition of the latency-rate server.

Consider the mth burst period of flow j. Let $\hat{a}^{j,k}$ denote the actual arrival time of the kth packet in the mth burst period of flow j at server $\mathcal{S}$. Then clearly, $\hat{a}^{j,1} = \tau$, where recall that τ is the beginning of the mth bursty period of flow j. For each packet $p^{j,k}$ in the mth burst period, define $\nu^{j,k}$ recursively as follows:

$$\nu^{j,1} = \hat{a}^{j,1} + \frac{L^{j,1}}{r^j} \text{ and } \nu^{j,k} = \max\{\nu^{j,k-1}, \hat{a}^{j,k}\} + \frac{L^{j,k}}{r^j} \; k \geq 2. \tag{4.37}$$

From the definition of burst period, it is not too hard to see that for $k \geq 1$, we must have $\hat{a}^{j,k} \leq \nu^{j,k-1}$. In other words, $\nu^{j,k} = \nu^{j,k-1} + \frac{L^{j,k}}{r^j}$. Consequently, we have

$$r^j(\nu^{j,k} - \hat{a}^{j,1}) = \sum_{q=1}^{k} L^{j,q}. \tag{4.38}$$

For each packet $p^{j,k}$, let $\hat{f}^{j,k}$ be the actual time it finishes service at server $\mathcal{S}$. The following lemma provides an alternative definition of a latency-rate server, the proof of which can be found in Appendix B.3.

Lemma 8 *A server $\mathcal{S}$ is a latency-rate server with a latency parameter Θ^j (with respect to flow j) if and only if for any packet $p^{j,k}$ of flow j, the following inequality holds:*

$$\hat{f}^{j,k} - \nu^{j,k} \leq \Theta^j - \frac{L^{j,k}}{r^j}.$$

■

Using Lemma 8, we now determine the error term for the latency rate server $\mathcal{S}$. For each packet $p^{j,k}$ of flow j, let $\tilde{\omega}^{j,k}$ be its virtual time as it enters $\mathcal{S}$. Define its virtual finish time $\tilde{\nu}^{j,k}$ by $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k} + \tilde{d}^{j,k}$. Using the fact that $\tilde{\omega}^{j,k} \geq \hat{a}^{j,k}$ and the virtual spacing property of $\tilde{\omega}^{j,k}$, it is not too hard to prove by induction that

$$\tilde{\nu}^{j,k} \geq \nu^{j,k}, \text{for all } k \geq 1.$$

Table 4.2: Error terms of latency-rate ($\mathcal{LR}$) servers.

$\mathcal{LR}$ server	PGPS/WFQ, VC, FFQ, SPFQ	SCFQ
Latency Θ^j	$\frac{L^{j,max}}{r^j} + \frac{L^{*,max}}{C}$	$\frac{L^{j,max}}{r^j} + \frac{L^{*,max}}{C}(N-1)$
Error term Ψ	$\frac{L^{*,max}}{C}$	$\frac{L^{*,max}}{C}(N-1)$

Then from Lemma 8, we have

$$\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \Theta^j - \frac{L^{j,k}}{r^j} \leq \tilde{\nu}^{j,k} + \Theta^j.$$

Hence we see that $\mathcal{S}$ has an error term Ψ such that $\Psi \leq \Theta^j$. This leads to the following theorem.

Theorem 12 *Any latency-rate server with a latency Θ^j (with respect to flow j) has an error term such that*

$$\Psi \leq \Theta^j.$$

■

For several well-known scheduling algorithms studied in [70, 72], we can actually show that $\Psi = \Theta^j - \frac{L^{j,max}}{r^j}$. Θ^j and its corresponding error term for these scheduling algorithms are listed in Table 4.2.

4.8 Discussions

4.8.1 Implementation Issues

So far we have focused on the theoretical foundation for the virtual time reference system. In this section we will briefly discuss issues regarding its implementation. Future work will further explore these issues.

A straightforward method to implement the virtual time reference system is to use the *dynamic packet state* (DPS) technique proposed in [73, 74]. Using this technique, the packet virtual time stamp is updated at every core router as a packet enters or departs. An

alternative method is to use *static packet state*. This technique requires an explicit path set-up procedure, which can be done, for example, using a simplified RSVP or MPLS (Multiprotocol Label Switching) [9, 66]. Note that this path set-up is *different* from a reservation set-up for a flow. In fact, multiple flows can share the same path. During the path set-up, the i^{th} router along the path is configured with a parameter $D_i = \sum_{q=1}^{i-1} \Psi_q + \sum_{q=1}^{i-1} \pi_{q,q+1}$, which represents the cumulative error term and propagation delay along the path up to router i. The i^{th} router is also configured with another parameter, p_i, which is the number of rate-based schedulers along the path up to router i (exclusive). Note that both D_i and p_i are parameters that are related only to the path characteristics, and are independent of any flow traversing the path. At the network edge, the virtual time stamp (i.e., $\tilde{\omega}_1^{j,k}$) of a packet is initialized to the time it is injected into the network core. At the i^{th} router along the path of its flow, the virtual time stamp associated with the packet is computed as $\tilde{\omega}_i^{j,k} = \tilde{\omega}_1^{j,k} + D_i + C_i$, where $C_i = p_i(\frac{L^{j,k}}{r^j} + \delta^{j,k}) + (i - 1 - p_i)d^j$. Using this approach, we see that once the packet state is initialized at the network edge, it will not be modified or updated inside the network core. This may speed up the packet forwarding operation of core routers. In particular, for a core router whose scheduling mechanism does not use the packet virtual time stamp information (e.g., a FIFO scheduler), there is no need to compute packet virtual time stamps.

The virtual time reference system is amenable to incremental deployment. Consider, for instance, the scenario where, between two core routers which employ the VT-EDF scheduling algorithms, there is one core router which does not support packet virtual time stamps. As long as the scheduling mechanism of this router can be characterized by an error term Ψ, its effect on the virtual time can be treated as if it were part of the link propagation delay between the two virtual-time-aware routers, and can be absorbed into the propagation delay between these two routers.

A critical question in implementing the virtual time reference system is how to encode the packet state. In its most general form, the packet state contains four parameters: packet virtual time stamp $\tilde{\omega}^{j,k}$, reserved rate r^j, delay parameter d^j and virtual time adjustment term $\delta^{j,k}$. Observe that in terms of providing end-to-end delay bounds, a rate-based scheduler can be treated as if it were a delay-based scheduler with a virtual delay parameter $L^{j,max}/r^j$ (see the comment at the end of Section 4.3.1). Hence we can eliminate the virtual time adjustment term $\delta^{j,k}$ completely. As discussed in [73], there are several options that we can use to encode the packet state: using an IP option, using an MPLS label, using the IP fragment offset field. Using the last option, for example, the packet state informa-

tion can be efficiently encoded using floating point representation with 17 bits [73]. Our virtual time reference system allows for additional flexibility in packet state encoding. In the case where only coarse-grain QoS guarantees (say, a fixed number of bandwidth or delay classes) is to be supported (e.g., in a DiffServ domain), the rate and delay parameters can be encoded in the TOS bits of the IP header as part of PHBs. Thus only the packet virtual time stamp needs to be carried in a separate field. It is possible to represent packet virtual time stamps *approximately*, using the notion of *slotted virtual time*. Accuracy of the approximation clearly hinges on the number of bits available to represent the virtual time. These issues will be investigated further in the future work.

4.8.2 QoS Provisioning and Admission Control

As stated in the introduction, one major objective of our work is to use the virtual time reference system as a QoS abstraction to decouple the data plane from the QoS control plane so as to facilitate the design of a bandwidth broker architecture for guaranteed services. In this section we briefly describe how this may be achieved. The details are left to future work.

In our bandwidth broker architecture, *no QoS reservation state, whether per-flow or aggregate reservation state, is maintained at any core router*. Bandwidth brokers have the topology information of a network domain and *dynamically* maintain all the QoS states regarding the flows and routers. When a request for setting QoS reservation for a flow arrives, a bandwidth broker looks up its QoS database, finds an appropriate path and checks whether sufficient resources are available at each router along the path. In case the flow can be admitted, the bandwidth broker would choose a reserved rate r and a delay parameter d for the flow, and informs the edge router to configure the edge conditioner accordingly. No QoS configuration or "state update" is needed at core routers *on a per-flow basis*. As a result, the problem of *robustness* facing the conventional hop-by-hop admission control approach [73], e.g., *inconsistent* QoS databases due to loss of signaling messages, is significantly alleviated. Because of the virtual time reference system, this bandwidth broker architecture is capable of supporting QoS provisioning and admission control with similar granularity and flexibility of the IntServ guaranteed service. The decoupling of QoS control plane and data plane enables sophisticated admission control algorithms to be employed in powerful bandwidth brokers for network-wide optimization of resource utilization. This is generally *infeasible* in the conventional hop-by-hop admission control approach. Furthermore, the bandwidth broker architecture makes it easy to implement policy-based admis-

sion control or advanced reservation.

4.9 Summary

In this chapter we have proposed and developed a novel virtual time reference system as a unifying scheduling framework to provide scalable support for guaranteed services. This virtual time reference system is designed as a conceptual framework upon which guaranteed services can be implemented in a scalable manner using the DiffServ paradigm. The key construct in the proposed virtual time reference system is the notion of packet virtual time stamp, whose computation is core stateless, i.e., no per-flow states are required for its computation. In the chapter, we have laid the theoretical foundation for the definition and construction of packet virtual time stamps. We described how per-hop behavior of a core router (or rather its scheduling mechanism) can be characterized via packet virtual time stamps, and based on this characterization, establish end-to-end per-flow delay bounds. Consequently, we demonstrated that, in terms of its ability to support guaranteed services, the proposed virtual time reference system has the same expressive power as the IntServ model. Furthermore, we showed that the notion of packet virtual time stamps leads to the design of new core stateless scheduling algorithms, especially work-conserving ones. In addition, our framework does not exclude the use of existing scheduling algorithms such as stateful fair queueing algorithms to support guaranteed services.

Part II

Scalable Network Resource Management Control Plane

Chapter 5

Background and Overview

The ability to provide end-to-end guaranteed services (e.g., guaranteed delay) for networked applications is a desirable feature of the future Internet. To enable such services, Quality-of-Service (QoS) support from *both the network data plane* (e.g. packet scheduling) *and the control plane* (e.g., admission control and resource reservation) is needed. In the first part of this dissertation, we have investigated how powerful and flexible QoS can be supported in the Internet in a scalable manner. In this part, we focus on reducing the operational complexity on the control plane.

Previous attempts at reducing the complexity of QoS control plane have mostly followed the conventional *hop-by-hop* reservation set-up approach adopted by RSVP and ATM through *QoS control state aggregation*. In the conventional hop-by-hop reservation set-up approach, the QoS reservation set-up request of a flow is passed from the ingress router towards the egress router along the path of the flow, where each router along the path processes the reservation set-up request and determines whether the request can be honored or not *by administering a local admission control test using its own QoS state information.* However, due to the distributed nature of this approach and unreliability of the network, potential inconsistency (e.g., due to loss of signaling messages) may result in the QoS states maintained by each router, which may cause serious problems in network QoS management. RSVP addresses this problem by using *soft states,* which requires routers to periodically retransmit PATH and RESV messages, thus incurring additional communication and processing overheads. These overheads can be reduced through a number of state reduction techniques [40, 79, 80]. Under the *core stateless* framework proposed in [73], the scalability issue of QoS control plane is addressed by maintaining only *aggregate reservation state*

at each router. The problem of inconsistent QoS states is tackled via a novel *bandwidth estimation* algorithm, which relies on the dynamic reservation information periodically carried in packets, and incurs additional processing overhead at core routers.

The conventional hop-by-hop reservation set-up approach ties such QoS control functions as admission control, resource reservation and QoS state management to core routers, whether per-flow or aggregate QoS states are maintained at core routers. Besides the issues discussed above, this approach requires admission control and QoS state management modules to be installed at every single router to support guaranteed services. As a result, if a new level of service (say, a new guaranteed delay service class) is introduced into a network, it may require upgrade or reconfiguration of the admission control modules at some or all core routers. An alternative, and perhaps more attractive, approach is the *bandwidth broker* (BB) architecture, which is first proposed in [58] for the *Premium Service* using the DiffServ model. Under this BB architecture, admission control, resource provisioning and other policy decisions are performed by a centralized bandwidth broker in each network domain.

This centralized bandwidth broker model for QoS control and management has several appealing features. For example, the centralized bandwidth broker model decouples (to a large extent) the QoS control plane from the data plane. In particular, QoS control functions such as admission control and QoS state maintenance are removed from the core routers of a network domain, reducing the complexity of the core routers. Consequently, no hop-by-hop signaling for reservation set-up along the data path is needed, removing the signaling overhead from core routers. Furthermore, because the network QoS states are centrally managed by the bandwidth broker, the problems of unreliable or inconsistent control states are circumvented [73]. This is in contrast to the IETF IntServ QoS control model based on RSVP [6, 84], where every router participates in hop-by-hop signaling for reserving resources and maintains its own QoS state database. Hence in this respect, the centralized bandwidth broker model provides a more scalable alternative for QoS control and management.

On the other hand, although several implementation efforts in building bandwidth brokers are under way (see, e.g., [75]), so far it is not clear what level of guaranteed services can be supported and whether core routers are still required to perform *local* admission control under the proposed BB architecture in [58]. Moreover, the centralized bandwidth broker model for QoS control and management also introduces its own scalability issue, in particular, the ability of the bandwidth broker to handle large volumes of flows as the network

system scales. In a DiffServ network where only slow time scale, static resource provisioning and traffic engineering (e.g., those performed to set up virtual private networks) are performed, the scalability problem may not be acute. But with the rapid evolution of today's Internet, many new applications and services such as Voice over IP (VoIP), on-demand media streaming and real-time content delivery (e.g., stock quotes and news) may require dynamic QoS control and management such as admission control and resource provisioning at the time scale of flow arrival and departure. In these circumstances, an inappropriately centralized bandwidth broker system can become a potential bottleneck, limiting the number of flows that can be accommodated into the network system, while the network system itself is still under-loaded. See [3, 76, 85] for more detailed discussions on the issues in designing and building such a centralized bandwidth broker architecture.

In this part, we will present two scalable bandwidth broker architectures. In Chapter 6, we design a centralized bandwidth broker architecture, which relies on the virtual time reference system we studies in the last chapter to completely decouple the QoS control plane from the packet forwarding data plane. Using this bandwidth broker architecture, we demonstrate how admission control can be done on an entire path basis, instead of on a "hop-by-hop" basis, which may significantly reduce the complexity of the admission control algorithms. In Chapter 7, we present a hierarchical bandwidth broker architecture to further improve the control plane scalability in supporting QoS in the Internet.

Chapter 6

A Centralized Bandwidth Broker Architecture

6.1 Introduction

In this chapter we present a novel bandwidth broker architecture for scalable support of guaranteed services that *decouples the QoS control plane from the packet forwarding plane*. More specifically, under this BB architecture, The QoS reservation states are stored at and managed solely by the bandwidth broker(s) in a network domain. Despite this fact, our bandwidth broker architecture is still *capable of providing end-to-end guaranteed services, whether fine-grain per-flow delay guarantees or coarse-grain class-based delay guarantees*. This bandwidth broker architecture is built upon the *virtual time reference system* developed in [86]. This virtual time reference system is designed as a *unifying* scheduling framework based on which both the *per-hop behaviors* of core routers (in terms of their abilities to provide delay and bandwidth guarantees) and the *end-to-end properties* of their concatenation can be characterized. Furthermore, it also provides a QoS abstraction for scheduling mechanisms of core routers that allows the bandwidth broker(s) in a network domain to perform (either per-flow or aggregate) QoS control functions such as admission control and reservation set-up with no or minimal assistance from core routers.

Because of this decoupling of data plane and QoS control plane, our bandwidth broker architecture is appealing in several aspects. First of all, by maintaining QoS reservation states only in a bandwidth broker (or bandwidth brokers), core routers are relieved of QoS control functions such as admission control, making them potentially more efficient. Second,

and perhaps more importantly, a QoS control plane that is decoupled from the data plane allows a network service provider to introduce new (guaranteed) services without necessarily requiring software/hardware upgrades at core routers. Third, with QoS reservation states maintained by a bandwidth broker, it can perform sophisticated QoS provisioning and admission control algorithms to optimize network utilization in a *network-wide* fashion. Such network-wide optimization is difficult, if not impossible, under the conventional hop-by-hop reservation set-up approach. Furthermore, the problem of inconsistent QoS states facing the hop-by-hop reservation set-up approach is also significantly alleviated under our approach. Last but not the least, under our approach, the reliability, robustness and scalability issues of QoS control plane (i.e., the bandwidth broker architecture) can be addressed *separately from, and without incurring additional complexity to, the data plane*, for example, by using distributed or hierarchical bandwidth brokers [88].

To illustrate some of the advantages presented above, in this chapter we will primarily focus on the design of efficient admission control under the proposed bandwidth broker architecture. We consider both *per-flow* end-to-end guaranteed delay services and *class-based* guaranteed delay services with flow aggregation. Using our bandwidth broker architecture, we demonstrate how admission control can be performed at an entire *path* level, instead of on a "hop-by-hop" basis. Such an approach *can* significantly reduce the complexity of the admission control algorithms. In designing class-based admission control algorithms, we investigate the problem of flow aggregation in providing guaranteed delay services, and devise a new apparatus to effectively circumvent this problem using our bandwidth broker architecture. We conduct extensive analysis to provide theoretical underpinning for our schemes as well as to establish their correctness. Simulations are also performed to demonstrate the efficacy of our schemes.

The remainder of this chapter is structured as follows. In Section 6.2, we first briefly review the virtual time reference system, and then present an overview of our proposed bandwidth broker architecture. In Section 6.3, we present per-flow path-oriented admission control algorithms. These admission control algorithms are extended in Section 6.4 to address class-based guaranteed delay services with flow aggregation. Simulation investigation is conducted in Section 6.5, and the chapter is concluded in Section 6.6.

6.2 Bandwidth Broker Architecture Overview

In this section we present a brief overview of a novel bandwidth broker architecture for scalable support of guaranteed services. This bandwidth broker architecture relies on the virtual time reference system, which we studied in the last chapter, to provide a QoS abstraction of the data plane. First, we need to provide an end-to-end delay bound within the virutal time reference system by incoporating the delays packet experienced at edge routers.

An important consequence of the virtual time reference system outlined above is that the end-to-end delay bound on the delay experienced by packets of a flow across the network core can be expressed in terms of the rate-delay parameter pair of a flow and the error terms of the routers along the flow's path. Suppose there are total h hops along the path of flow j, of which q routers employ rate-based schedulers, and $h-q$ delay-based schedulers. Then for each packet $p^{j,k}$ of flow j, we have

$$\hat{f}_h^{j,k} - \hat{a}_1^{j,k} \leq d_{core}^j = q\frac{L^{j,max}}{r^j} + (h-q)d^j + \sum_{i=1}^{h}\Psi_i + \sum_{i=1}^{h-1}\pi_i \tag{6.1}$$

where $L^{j,max}$ is the maximum packet size of flow j.

Suppose the traffic profile of flow j is specified using the standard dual-token bucket regulator $(\sigma^j, \rho^j, P^j, L^{j,max})$ where $\sigma^j \geq L^{j,max}$ is the maximum burst size of flow j, ρ^j is the sustained rate of flow j, P^j is the peak rate of flow j. Then the maximum delay packets of flow j experienced at the edge shaper is bounded by

$$d_{edge}^j = T_{on}^j\frac{P^j - r^j}{r^j} + \frac{L^{j,max}}{r^j} \tag{6.2}$$

where $T_{on}^j = (\sigma^j - L^{j,max})/(P^j - \rho^j)$ is the maximum duration that flow j can inject traffic at its peak rate into the network (here the edge traffic conditioner). Hence the end-to-end delay bound for flow j is given by

$$d_{end-to-end}^j = d_{edge}^j + d_{core}^j = T_{on}^j\frac{P^j - r^j}{r^j} + (q+1)\frac{L^{j,max}}{r^j} + (h-q)d^j + \sum_{i=1}^{h}\Psi_i + \sum_{i=1}^{h-1}\pi_i. \tag{6.3}$$

Observe that the end-to-end delay formula is quite similar to that specified in the IETF Guaranteed Service using the WFQ as the reference system. In this sense, the virtual time

reference system provides a conceptual *core stateless* framework based on which guaranteed services can be implemented in a scalable manner using the DiffServ paradigm.

Now let's move to the overview of the bandwidth broker architecture. Each router[1] in the network domain is characterized by an error term. The novelty of the proposed bandwidth broker architecture lies in that *all QoS reservation and other QoS control state information (e.g., the amount of bandwidth reserved at a core router) is removed from core routers, and is solely maintained at and managed by bandwidth broker(s).* In supporting guaranteed services in the network domain, core routers perform *no QoS control and management functions* such as admission control, but only *data plane functions* such as packet scheduling[2] and forwarding. In other words, the data plane of the network domain is decoupled from the QoS control plane. Despite the fact that all the QoS reservation states are removed from core routers and maintained solely at the bandwidth broker, the proposed bandwidth broker architecture is capable of supporting guaranteed services with the same granularity and expressive power (if not more) as the IntServ/Guaranteed Service model. This is achieved without the potential complexity and scalability problems of the IntServ model. Furthermore, as mentioned in the introduction, our bandwidth broker architecture for supporting guaranteed services has many appealing features that makes it more *flexible*, and arguably, more *scalable*. In this chapter, we will illustrate some of these advantages by addressing the admission control problem under the proposed bandwidth broker architecture. The major components of the bandwidth broker architecture (in particular, those pertinent to the admission control) are described below.

As shown in Figure 6.1, the bandwidth broker (BB)[3] consists of several modules such as admission control, QoS routing and policy control. In this chapter, we will focus primarily on the admission control module. The BB also maintains a number of management

[1]The bandwidth broker architecture presented in this chapter is designed for supporting guaranteed services only. We assume that each router deploys an appropriate (i.e., characterizable by the notion of error term) scheduler with certain amount of bandwidth and buffer size provisioned for supporting guaranteed delay service. Hence precisely speaking, the error term of a router is that of the scheduler deployed at the router. Throughout the chapter, by a router we mean the scheduler of the router used for supporting guaranteed services.

[2]Since the virtual time reference system does not mandate the scheduling mechanism used by a core router, in particular, that core stateless schedulers such as $C_{\mathcal{S}}VC$ and VT-EDF be used, it is possible for some routers (e.g., those at or near the network edge) to implement stateful scheduling algorithms (e.g., static WFQ or even per-flow WFQ [86]) to support guaranteed services. In the latter case, certain *scheduling* parameters (either class-based or per-flow) may need to statically or dynamically configured by the bandwidth broker, e.g., during a path set-up or at other occasions.

[3]For simplicity, we assume that there is a single centralized BB for a network domain. In practice, there can be multiple BBs for a network domain to improve reliability and scalability [88].

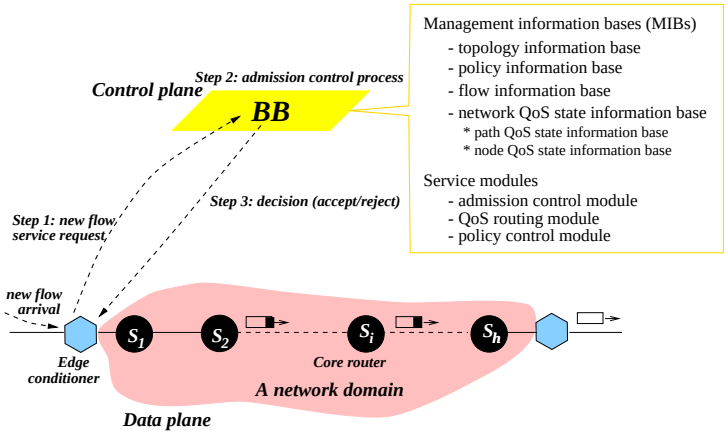


Figure 6.1: Illustration of a bandwidth broker (BB) and its operation in a VTRS network domain.

information bases (MIB) for the purpose of QoS control and management of the network domain. For example, the *topology information base* contains topology information that the BB uses for route selection and other management and operation purposes; and the *policy information base* contains policies and other administrative regulations of the network domain. In the following we describe the MIBs that are used by the admission control module.

Flow Information Base. This MIB contains information regarding individual flows such as flow id., traffic profile (e.g., $(\sigma^j, \rho^j, P^j, L^{j,max})$), service profile (e.g., end-to-end delay requirement D^j), route id. (which identifies path that a flow traverses in the network domain) and QoS reservation ($\langle r^j, d^j \rangle$ in the case of per-flow guaranteed delay services, or a delay service class id. in the case of class-based guaranteed services) associated with each flow. Other administrative (e.g., policy, billing) information pertinent to a flow may also be maintained here.

Network QoS State Information Bases. These MIBs maintain the QoS states of the network domain, and thus are the key to the QoS control and management of the network domain. Under our BB architecture, the network QoS state information is represented in two-levels using two separate MIBs: *path QoS state information base* and *node QoS state information base*. These two MIBs are presented in detail below.

Path QoS state information base maintains a set of paths (each with a route id.) between various ingress and egress routers of the network domain. These paths can

be pre-configured or dynamically set up[4]. Associated with each path are certain static parameters characterizing the path and dynamic QoS state information regarding the path. Examples of static parameters associated a path $\mathcal{P}$ are the number of hops h on $\mathcal{P}$, the number of rate-based schedulers (q) and delay-based schedulers $(h-q)$ along $\mathcal{P}$, sum of the router error terms and propagation delay along $\mathcal{P}$, $D_{tot}^{\mathcal{P}} = \sum_{i \in \mathcal{P}}(\Psi_i + \pi_i)$, and the maximum permissible packet size (i.e., MTU) $L^{\mathcal{P},max}$. The dynamic QoS state information associated with $\mathcal{P}$ include, among others, the set of flows traversing $\mathcal{P}$ (in the case of per-flow guaranteed delay services) or the set of delay service classes and their aggregate traffic profiles (in the case of class-based guaranteed delay services) and a number of QoS state parameters regarding the (current) QoS reservation status of $\mathcal{P}$ such as the minimal remaining bandwidth $C_{res}^{\mathcal{P}}$ along $\mathcal{P}$, a sorted list of delay parameters currently supported along $\mathcal{P}$ and associated minimal residual service points, and the set of "bottleneck" nodes along $\mathcal{P}$.

Node QoS state information base maintains information regarding the routers in the network domain. Associated with each router is a set of static parameters characterizing the router and a set of dynamic parameters representing the router's current QoS state. Examples of static parameters associated a router $\mathcal{S}$ are the scheduler type(s) (i.e., rate- or delay-based), its error term(s) Ψ, propagation delays to its next-hop routers π's, configured total bandwidth C and buffer size B for supporting guaranteed delay services, and if applicable, a set of delay classes and their associated delay parameters, and/or a set of pre-provisioned bandwidth and buffer size pairs $\langle C_k, B_k \rangle$ for each delay class supported by $\mathcal{S}$. The dynamic router QoS state parameters include the current residual bandwidth $C_{res}^{\mathcal{S}}$ at $\mathcal{S}$, a sorted list of delay parameters associated with flows traversing $\mathcal{S}$ and their associated minimal residual service points at $\mathcal{S}$, and so forth.

In the following sections we will illustrate how some of the path and router parameters will be utilized and maintained by the BB to perform efficient admission control. Before we move to the problem of admission control using the proposed BB architecture, we briefly discuss the basic operations of the BB, in particular, those pertinent to the *admission control module*.

[4]Note that during the process of a path set-up, no admission control test is administered. The major function of the path set-up process is to configure forwarding tables of the routers along the path, and if necessary, provision certain scheduling/queue management parameters at the routers, depending on the scheduling and queue management mechanisms deployed. Hence we refer to such a path a *traffic engineered* (TE) path. Set-up of such a TE path can be done by using a path set-up signaling protocol, say, MPLS [9, 66], or a simplified version (minus resource reservation) of RSVP.

When a new flow with traffic profile $(\sigma^j, \rho^j, P^j, L^{j,max})$ and end-to-end delay requirement $D^{j,req}$ arrives at an ingress router, the ingress router sends a new flow service request message to the BB. Upon receiving the service request, the BB first checks for policy and other administrative information bases to determine whether the new flow is admissible. If not, the request is immediately rejected. Otherwise, the BB selects a path[5] (from the ingress to an appropriate egress router in the network domain) for the new flow, based on the network topology information and the current network QoS state information, in addition to other relevant information (such as policy constraints applicable to this flow).

Once the path is selected, the BB will invoke the admission control module to determine if the new flow can be admitted. The details of admission control procedure for supporting per-flow guaranteed delay services and class-based guaranteed delay services will be presented in Section 6.3 and Section 6.4, respectively. Generally speaking, the admission control procedure consists of two phases: 1) *admission control test* phase during which it is determined whether the new flow service request can be accommodated and how much network resources must be reserved if it can be accommodated; and 2) *bookkeeping* phase during which the relevant information bases such as the flow information base, path QoS state information base and node QoS state information base will be updated, if the flow is admitted. If the admission control test fails, the new flow service request will be rejected, no information bases will be updated. In either case, the BB will inform the ingress of the decision. In the case that the new flow service request is granted, the BB will also pass the QoS reservation information (e.g., $\langle r^j, d^j \rangle$) to the ingress router so that it can set up a new or re-configure an existing edge conditioner (which is assumed to be co-located at the ingress router) for the new flow. The edge conditioner will appropriately initialize and insert the packet states into packets of the new flow once it starts to send packets into the network.

Before we proceed to discuss admission control algorithms in the proposed bandwidth broker architecture, it is worth noting that the proposed bandwidth broker architecture is only a *conceptually centralized* architecture. It can be implemented in a distributed matter in practice. In particular, the separation of *Path QoS State Information Base* and *Node QoS State Information Base* facilitates the design of distributed (hierarchical) bandwidth broker

[5]If necessary, a new path may be set up dynamically. The problem of QoS routing, i.e., finding an "optimal" path for the flow can be handled relatively easily under our BB architecture. Since it is beyond the scope of this chapter, we will not discuss it here. As an aside, our BB architecture can also accommodate advance QoS reservation in a fairly straightforward fashion, owing to decoupling of the QoS control plane and data plane and the centralized network QoS state information bases.

systems [88]. Moreover, the admission control algorithms developed in this chapter can be used as middlewares to build distributed bandwidth broker systems.

6.3 Admission Control for Per-Flow Guaranteed Services

In this section, we study the problem of admission control for support of per-flow guaranteed services under the proposed bandwidth broker architecture. We present a *path-oriented* approach to perform efficient admission control test and resource allocation. Unlike the conventional *hop-by-hop* approach which performs admission control *individually* based on the *local QoS state* at each router along a path, this path-oriented approach examines the resource constraints *along the entire path simultaneously*, and makes admission control decision accordingly. As a result, we can significantly reduce the time of conducting admission control test. Furthermore, we can also perform path-wide optimization when determining resource allocation for a new flow. Clearly, such a path-oriented approach is possible because the availability of QoS state information of the entire path at the bandwidth broker.

6.3.1 Path with Only Rate-based Schedulers

To illustrate how the path-oriented approach works, we first consider a simple case, where we assume that the path $\mathcal{P}$ for a new flow ν consists of only rate-based schedulers. Hence in this case, we only need to determine whether a reserved rate r^ν can be found for the new flow for it to be admitted. The delay parameter d^ν will not be used. For simplicity of exposition, we assume that a scheduler such as *core-stateless virtual clock* ($C_\not{S}$VC) or *core-jitter virtual clock* (CJVC) is employed at the routers $\mathcal{S}_i$ along $\mathcal{P}$. Let $j \in \mathcal{F}_i$ denote that flow j currently traverses $\mathcal{S}_i$, and C_i be the total bandwidth at $\mathcal{S}_i$. Then as long as $\sum_{j\in\mathcal{F}_i} r^j \leq C_i$, $\mathcal{S}_i$ can guarantee each flow j its reserved bandwidth r^j. We use $C_{res}^{\mathcal{S}_i}$ to denote the residual bandwidth at $\mathcal{S}_i$, i.e., $C_{res}^{\mathcal{S}_i} = C_i - \sum_{j\in\mathcal{F}_i} r^j$. We consider the two phases of the admission control procedure.

Admission Test. Let $(\sigma^\nu, \rho^\nu, P^\nu, L^{\nu,max})$ be the traffic profile of a new flow ν, and $D^{\nu,req}$ be its end-to-end delay requirement. Let h be the number of hops in $\mathcal{P}$, the path for the new flow. From (6.3), in order to meet its end-to-end delay requirement $D^{\nu,req}$, the reserved rate r^ν for the new flow ν must satisfy: 1) $\rho^\nu \leq r^\nu \leq P^\nu$, and 2)

$$D^{\nu,req} \geq d_{edge}^\nu + d_{core}^\nu = T_{on}^\nu \frac{P^\nu - r^\nu}{r^\nu} + (h+1)\frac{L^{\nu,max}}{r^\nu} + D_{tot}^{\mathcal{P}} \tag{6.4}$$

where $T_{on}^{\nu} = (\sigma^{\nu} - L^{\nu,max})/(P^{\nu} - \rho^{\nu})$ and $D_{tot}^{\mathcal{P}} = \sum_{i \in \mathcal{P}} \Psi_i + \sum_{i \in \mathcal{P}, i \neq h} \pi_i$.

Furthermore, r^{ν} must not exceed the minimal residual bandwidth $C_{res}^{\mathcal{P}}$ along path $\mathcal{P}$, where $C_{res}^{\mathcal{P}} = \min_{i \in \mathcal{P}} C_{res}^{\mathcal{S}_i}$ is maintained, as a path QoS parameter associated with $\mathcal{P}$, in the path QoS state MIB.

Let r_{min}^{ν} be the smallest r^{ν} that satisfies (6.4), i.e., $r_{min}^{\nu} = [T_{on}^{\nu} P^{\nu} + (h+1)L^{\nu,max}]/[D^{\nu,req} - D_{tot}^{\mathcal{P}} + T_{on}^{\nu}]$. Define

$$r_{fea}^{low} = \max\{\rho^{\nu}, r_{min}^{\nu}\} \text{ and } r_{fea}^{up} = \min\{P^{\nu}, C_{res}^{\mathcal{P}}\}.$$

Then $\mathcal{R}_{fea}^{*} = [r_{fea}^{low}, r_{fea}^{up}]$ is the *feasible rate range*, from which a feasible reserved rate r^{ν} can be selected. Clearly, if $\mathcal{R}_{fea}^{*}$ is empty, then the service request of the new flow ν must be rejected. Otherwise, it is admissible, and $r^{\nu} = r_{fea}^{low}$ is the *minimal* feasible reserved rate for the new flow ν. Given that the path QoS parameters $D_{tot}^{\mathcal{P}}$ and $C_{res}^{\mathcal{P}}$ associated with $\mathcal{P}$ are maintained in the path QoS state MIB, the above admission test can be done in $O(1)$.

Bookkeeping. If the new flow ν is admitted into the network, several MIBs (e.g., the flow MIB, the path and node QoS state MIBs) must be updated. The flow id., traffic profile and service profile of the new flow will be inserted into the flow MIB. The minimal residual bandwidth $C_{res}^{\mathcal{P}}$ will be subtracted by r^{ν}, the reserved rate for flow ν. Similarly, for each $\mathcal{S}_i$ along $\mathcal{P}$, its residual bandwidth $C_{res}^{\mathcal{S}_i}$ will also be subtracted by r^{ν}. Furthermore, for any path $\mathcal{P}'$ that traverses $\mathcal{S}_i$, its minimal residual bandwidth $C_{res}^{\mathcal{P}'}$ may also be updated, depending on whether the update of $C_{res}^{\mathcal{S}_i}$ changes $C_{res}^{\mathcal{P}'}$. Lastly, note that when an existing flow departs the network, the relevant MIBs should also be updated.

6.3.2 Path with Mixed Rate- and Delay-based Schedulers

We now consider the general case where the path $\mathcal{P}$ for a new flow ν consists of both rate-based and delay-based schedulers. In this case, we need to determine whether a rate-delay parameter pair $\langle r^{\nu}, d^{\nu} \rangle$ can be found for the new flow ν for it to be admitted. Let q be the number of rate-based schedulers and $h - q$ the number of delay-based schedulers along path $\mathcal{P}$. For simplicity of exposition, we assume[6] that the rate-based schedulers $\mathcal{S}_i$ along path $\mathcal{P}$ employ $C_{\not s}$VC (or any similar) scheduling algorithm whose schedulability condition is $\sum_{j \in \mathcal{F}_i} r^j \leq C_i$, whereas the delay-based schedulers $\mathcal{S}_i$ employ the *VT-EDF*

[6]The schedulability conditions of $C_{\not s}VC$ and VT-EDF scheduling algorithms are representative of rate-based and delay-based scheduling algorithms [86]. Hence the results presented here for these two schedulers are also applicable to other schedulers, perhaps with some modifications.

scheduling algorithm, whose schedulability condition is given in (4.33). Hence if $\mathcal{S}_i$ is a rate-based scheduler along $\mathcal{P}$, it can guarantee each flow j its reserved bandwidth r^j, as long as $\sum_{j\in\mathcal{F}_i} r^j \leq C_i$. Similarly, if $\mathcal{S}_i$ is a delay-based scheduler along $\mathcal{P}$, it can guarantee each flow j its delay parameter d^j, as long as the schedulability condition (4.33) is satisfied. We now consider the two phases of the admission control procedure.

Admission Test. Because of the inter-dependence of the reserved rate r^ν and the delay parameter d^ν in the end-to-end delay bound (6.3) as well as the more complex schedulability condition (4.33) for the delay-based schedulers, the admission test for this case is less straightforward. By uncovering the monotonicity properties of the end-to-end delay formula (6.3) and schedulability condition (4.33), we show how an efficient admission test can be designed using the path-oriented approach. In addition, if the new flow ν is admissible, this admission test finds an *optimal* feasible rate-delay parameter pair $\langle r^\nu, d^\nu\rangle$ in the sense that r^ν is the *minimal* feasible rate. In other words, no other rate-delay parameter pair $\langle r^{\nu'}, d^{\nu'}\rangle$ such that $r^{\nu'} < r^\nu$ is feasible.

Before we present the algorithm, we need to introduce some notation and transform the end-to-end delay formula (6.3) as well as the schedulability condition (4.33) into a form such that their monotonicity properties can be derived. As before, let $(\sigma^\nu, \rho^\nu, P^\nu, L^{\nu,max})$ be the traffic profile of the new flow ν, and $D^{\nu,req}$ its end-to-end delay requirement. In order for the new flow ν to be admitted along the path $\mathcal{P}$ with a rate-delay parameter pair $\langle r^\nu, d^\nu\rangle$, its end-to-end delay requirement $D^{\nu,req}$ must be satisfied, namely, 1) $\rho^\nu \leq r^\nu \leq P^\nu$, and

$$D^{\nu,req} \geq d^\nu_{edge} + d^\nu_{core} = T^\nu_{on}\frac{P^\nu - r^\nu}{r^\nu} + (q+1)\frac{L^{\nu,max}}{r^\nu} + (h-q)d^\nu + D^{\mathcal{P}}_{tot}. \quad (6.5)$$

Furthermore, the schedulability condition at each scheduler $\mathcal{S}_i$ must not be violated. Let $C^{\mathcal{P}}_{res}$ be the minimal residual bandwidth along $\mathcal{P}$, i.e., $C^{\mathcal{P}}_{res} = \min_{i\in\mathcal{P}} C^{\mathcal{S}_i}_{res}$. Then from the schedulability conditions for the rate- and delay-based schedulers, we see that $r^\nu \leq C^{\mathcal{P}}_{res}$. Furthermore, for every delay-based scheduler $\mathcal{S}_i$ along $\mathcal{P}$, let $\langle r^k_i, d^k_i\rangle$ be the rate-delay parameter pair of flow k, where $k \in \mathcal{F}_i$. Then for each $k \in \mathcal{F}_i$, $\mathcal{S}_i \in \mathcal{P}$ such that $d^k_i \geq d^\nu$, we must have

$$\sum_{\{j\in\mathcal{F}_i : d^j_i \leq d^k_i\}} [r^j(d^k_i - d^j_i) + L^{j,max}] + [r^\nu(d^k_i - d^\nu) + L^{\nu,max}] \leq C_i d^k_i. \quad (6.6)$$

In summary, in order for $\langle r^\nu, d^\nu\rangle$ to be a feasible rate-delay parameter pair for the new flow ν, we must have that $r^\nu \in [\rho^\nu, \min\{P^\nu, C^{\mathcal{P}}_{res}\}]$ and that r^ν and d^ν must satisfy (6.5) and

(6.6). We now transform (6.5) and (6.6) into simple constraints on r^ν that are functions of d^ν.

Define $t^\nu = \frac{1}{h-q}(D^{\nu,req} - D^{\mathcal{P}}_{tot} + T^\nu_{on})$ and $\Xi^\nu = \frac{1}{h-q}[T^\nu_{on}P^\nu + (q+1)L^{\nu,max}]$. After some simple algebraic manipulations, we can rewrite (6.5) in the following form:

$$d^\nu \leq t^\nu - \frac{\Xi^\nu}{r^\nu} \tag{6.7}$$

or equivalently,

$$r^\nu \geq \frac{\Xi^\nu}{t^\nu - d^\nu}. \tag{6.8}$$

Note that from (6.7), it is clear that $d^\nu \leq t^\nu$. Furthermore, if d^ν decreases, the upper bound on r^ν in (6.8) also decreases. Hence the feasible range for r^ν shrinks from the right, as d^ν decreases.

We now consider the delay constraints (6.6). Given any flow k traversing a delay-based scheduler $\mathcal{S}_i$ such that $d^k_i \geq d^\nu$, define

$$S^k_i = C_i d^k_i - \sum_{\{j \in \mathcal{F}_i : d^j_i \leq d^k_i\}} [r^j(d^k_i - d^j_i) + L^{j,max}]. \tag{6.9}$$

Then (6.6) becomes

$$r^\nu(d^k_i - d^\nu) + L^{\nu,max} \leq S^k_i. \tag{6.10}$$

Note that S^k_i denotes the *minimum residual service* over any time interval of length d^k_i at scheduler $\mathcal{S}_i$. Hence (6.10) states that the new flow ν can be accommodated at $\mathcal{S}_i$ with a rate-delay parameter pair $\langle r^\nu, d^\nu \rangle$ while *without* affecting the delay guarantee for flow k, if the service required by the new flow ν over any time interval of length d^k_i does not exceed S^k_i. For simplicity, we shall refer to S^k_i as the *minimum residual service* of $\mathcal{S}_i$ at time d^k_i.

We can consolidate the delay constraints at all the delay-schedulers along $\mathcal{P}$ as follows. Let $\mathcal{F}^{del}$ be the union of the sets of the flows at all the delay-based schedulers, i.e., $\mathcal{F}^{del} = \cup\{j \in \mathcal{F}_i : \mathcal{S}_i \text{ is delay-based}\}$. Suppose there are a total of M *distinctive* delay parameters

associated with the flows in $\mathcal{F}^{del}$. Let these distinctive M delay parameters be denoted by $d^1, d^2, \ldots, d^M$, where $0 \leq d^1 < d^2 < \cdots < d^M$. For $m = 1, 2, \ldots, M$, define

$$S^m = \min\{S_i^k : k \in \mathcal{F}_i \text{ and } d_i^k = d^m, \mathcal{S}_i \text{ is delay-based}\}. \tag{6.11}$$

Clearly, S^m denotes the minimal residual service *among all the delay-based schedulers* at time d^m. Hence we refer to S^m as the *minimal residual service of path $\mathcal{P}$ at time d^m*. With this notation, we see that the new flow ν can be accommodated along path $\mathcal{P}$ with a rate-delay parameter pair $\langle r^\nu, d^\nu \rangle$ while *without* affecting the delay guarantee for any flow at any delay-based scheduler along the path, if for any $d^m \geq d^\nu$, we have

$$r^\nu(d^m - d^\nu) + L^{\nu,max} \leq S^m.$$

Using (6.8), we can re-write the above inequality as the following constraint on r^ν:

$$r^\nu(d^m - t^\nu) \leq S^m - \Xi^\nu - L^{\nu,max}. \tag{6.12}$$

We now show how to use (6.8) and (6.12) to determine whether a feasible rate-delay parameter pair $\langle r^\nu, d^\nu \rangle$ exists, and if it exists, how to find the minimum feasible r^ν. Define $d^0 = 0$ and $d^{M+1} = \infty$. Then if the new flow ν is admissible, there must exist a rate-delay parameter pair $\langle r^\nu, d^\nu \rangle$, where $d^\nu \in [d^{m-1}, d^m)$ for some $m = 1, 2, \ldots, M+1$. From (6.7), it is clear that $0 \leq d^\nu \leq t^\nu$. Let m^* be such that $d^{m^*-1} < t^\nu \leq d^{m^*}$. Clearly, $[d^{m^*-1}, d^{m^*})$ is the rightmost delay interval that may contain a feasible d^ν. We now examine *iteratively* the validity of each of the delay intervals $[d^{m-1}, d^m)$, starting from $m = m^*$ down to $m = 1$.

For $m = m^*, m^* - 1, \ldots, 2, 1$, suppose $d^\nu \in [d^{m-1}, d^m)$. Then from (6.8) as well as the constraint that $r^\nu \in [\rho^\nu, \min\{P^\nu, C_{res}^{\mathcal{P}}\}]$, we must have $r^\nu \in \mathcal{R}_{fea}^m = [r_{fea}^{m,l}, r_{fea}^{m,r}]$, where

$$r_{fea}^{m,l} = \max\{\frac{\Xi^\nu}{t^\nu - d^{m-1}}, \rho^\nu\}, \quad r_{fea}^{m,r} = \min\{\frac{\Xi^\nu}{t^\nu - d^m}, P^\nu, C_{res}^{\mathcal{P}}\}. \tag{6.13}$$

Similarly, from (6.12), it is not too hard to see that $d^\nu \in [d^{m-1}, d^m)$ implies that $r^\nu \in \mathcal{R}_{del}^m = [r_{del}^{m,l}, r_{del}^{m,r}]$, where

$$\begin{aligned} r_{del}^{m,l} &= \max_{m \leq k < m^*} \{\frac{S^k - \Xi^\nu - L^{\nu,max}}{d^k - t^\nu}\}, \\ r_{del}^{m,r} &= \min\left\{\min_{m \leq k < m^*} \{\frac{\Xi^\nu + L^{\nu,max}}{t^\nu - d^k}\}, \min_{k \geq m^*} \frac{S^k - \Xi^\nu - L^{\nu,max}}{d^k - t^\nu}\right\}. \end{aligned} \tag{6.14}$$

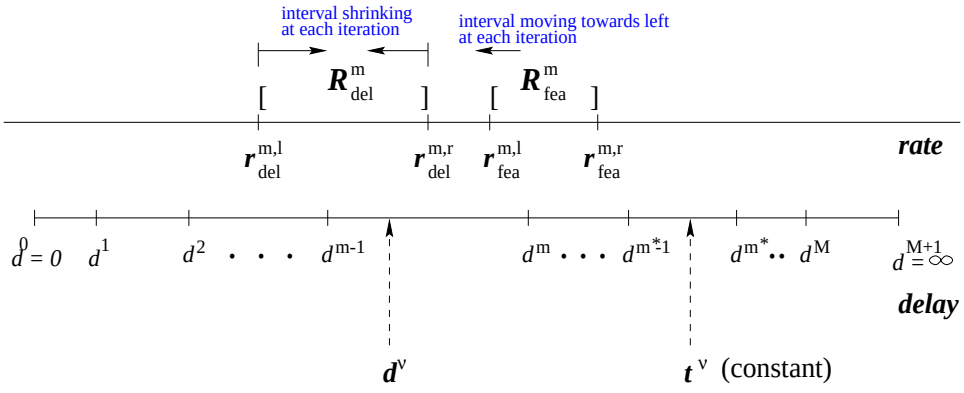


Figure 6.2: The behavior of feasible range $\mathcal{R}_{fea}^m$ and delay constraint range $\mathcal{R}_{del}^m$ at the mth iteration in the search of feasible rate-delay parameter pair $\langle r^\nu, d^\nu \rangle$ for a new flow ν.

Observe that when we move from the current delay interval $[d^{m-1}, d^m)$ to the next delay interval to the left $[d^{m-2}, d^{m-1})$, the corresponding feasible rate range $\mathcal{R}_{fea}^m$ determined by (6.13) also shifts to the left (see Figure 6.2). In contrast, the corresponding feasible rate range $\mathcal{R}_{del}^m$ determined by the delay constraints (6.14) shrinks: the left edge $r_{del}^{m,l}$ increases while the right edge $r_{del}^{m,r}$ decreases (see Figure 6.2). Using these monotonicity properties of $\mathcal{R}_{fea}^m$ and $\mathcal{R}_{del}^m$, we obtain the following theorem, which states whether a feasible rate-delay pair $\langle r^\nu, d^\nu \rangle$ exists such that $d^\nu \in [d^{m-1}, d^m]$. In addition, it also specifies whether the intervals to the left *may* contain a feasible solution, or a feasible solution with a smaller r^ν *may* exist.

Theorem 13 *If $\mathcal{R}_{fea}^m \cap \mathcal{R}_{del}^m$ is empty, then no feasible rate-delay pairs $\langle r^\nu, d^\nu \rangle$ exist such that $d^\nu \in [d^{m-1}, d^m]$. Furthermore, if $\mathcal{R}_{fea}^m$ is empty, or $\mathcal{R}_{del}^m$ is empty, or $r_{fea}^{m,r} < r_{del}^{m,l}$, then no intervals to the left contain a feasible solution either. More precisely, no feasible rate-delay pairs $\langle r^\nu, d^\nu \rangle$ exist such that $d^\nu \in [0, d^m)$.*

If $\mathcal{R}_{fea}^m \cap \mathcal{R}_{del}^m$ is not empty, then a feasible rate-delay pair $\langle r^\nu, d^\nu \rangle$ exists such that $d^\nu \in [d^{m-1}, d^m]$. Furthermore, if $r_{fea}^{m,l} < r_{del}^{m,l}$, then $r^\nu = r_{del}^{m,l}$ is the smallest rate such that there exists some $d^\nu \geq 0$ for which $\langle r^\nu, d^\nu \rangle$ is a feasible rate-delay pair. In other words, any rate-delay pair $\langle r^\nu, d^\nu \rangle$ where $r^\nu < r_{del}^{m,l}$ is not feasible.

Based on Theorem 13, the admission test is presented (in pseudo-code) in Figure 6.3. Starting with the rightmost interval $[d^{m^*-1}, d^{m^*})$, this algorithm determines iteratively whether a feasible rate-delay pair $\langle r^\nu, d^\nu \rangle$ exists such that $d^\nu \in [d^{m^*-1}, d^{m^*})$. If the new flow ν is admissible, the algorithm also finds the *feasible* rate-delay parameter pair $\langle r^\nu, d^\nu \rangle$ such that

0. $t^\nu = \frac{1}{h-q}[D^{\nu,req} - D^{\mathcal{P}}_{tot} + T^\nu_{on}]$
1. Let m^* such that $d^{m^*-1} < t^\nu \leq d^{m^*}$
2. **for** $m = m^*, m^* - 1, \ldots, 2, 1$
3. $\quad \mathcal{R}^m_{fea} \leftarrow [r^{m,l}_{fea}, r^{m,r}_{fea}]$
4. $\quad \mathcal{R}^m_{del} \leftarrow [r^{m,l}_{del}, r^{m,r}_{del}]$
5. $\quad$ **if** $(\mathcal{R}^m_{fea} \cap \mathcal{R}^m_{del} == \emptyset)$
6. $\quad\quad$ **if** $(\mathcal{R}^m_{fea} == \emptyset || \mathcal{R}^m_{del} == \emptyset || r^{m,r}_{fea} < r^{m,l}_{del})$
7. $\quad\quad\quad$ **break** with $d^\nu = d^m$
8. $\quad$ **else** /*$\mathcal{R}^m_{fea} \cap \mathcal{R}^m_{del} \neq \emptyset$*/
9. $\quad\quad$ **if** $(r^{m,l}_{fea} < r^{m,l}_{del})$
10. $\quad\quad\quad r^\nu \leftarrow r^{m,l}_{del}$
11. $\quad\quad\quad d^\nu \leftarrow t^\nu - \frac{\Xi^\nu}{r^\nu}$
12. $\quad\quad\quad$ **break** with d^ν
13. **if** $(d^\nu > t^\nu)$ no feasible value found
14. **else return** d^ν

Figure 6.3: Admission test for a new flow ν on a path with mixed rate- and delay-based schedulers.

r^ν is *minimal*. The time complexity of the algorithm is $O(M)$. Note that in general, we have $M \leq |\mathcal{F}^{del}| \leq \sum_{\mathcal{S}_i \text{ is delay-based}} |\mathcal{F}_i|$. Hence the complexity of the algorithm hinges only on the number of distinctive delay parameters supported by the schedulers along the path of the new flow. This reduction in complexity can be significant if many flows have the same delay requirements. This is particularly the case when we consider class-based admission control with flow aggregation where a number of fixed delay classes are pre-defined. Clearly this reduction in complexity is achieved because our admission control algorithm considers all the admissibility constraints along a path simultaneously. This is *not* possible using the conventional hop-by-hop reservation set-up approach (e.g., as employed in the IETF IntServ model with RSVP).

Bookkeeping. When a new flow is admitted in the network, the BB needs to update the flow MIB, path QoS state MIB and node QoS state MIB, among others. For a path $\mathcal{P}$ with mixed rate-based and delay-based schedulers, in addition to path parameters such as $D^{\mathcal{P}}_{tot}$ and $C^{\mathcal{P}}_{res}$, we assume that the minimum residual service S^m at each d^m is also maintained, where d^m is a distinctive delay parameter supported by one of the delay-based schedulers along $\mathcal{P}$. These parameters facilitate the admission test described in Figure 6.3. Hence when a new flow is admitted along path $\mathcal{P}$, these parameters need also to be updated. Furthermore, we assume that for each delay-based scheduler $\mathcal{S}_i$, the minimum residual service S^k_i of $\mathcal{S}_i$ at each d^k_i is also maintained at the node QoS state MIB. Hence these parameters must be updated accordingly. Furthermore, for any path traversing $\mathcal{S}_i$, the corresponding path minimum residual service parameters may also need to be updated.

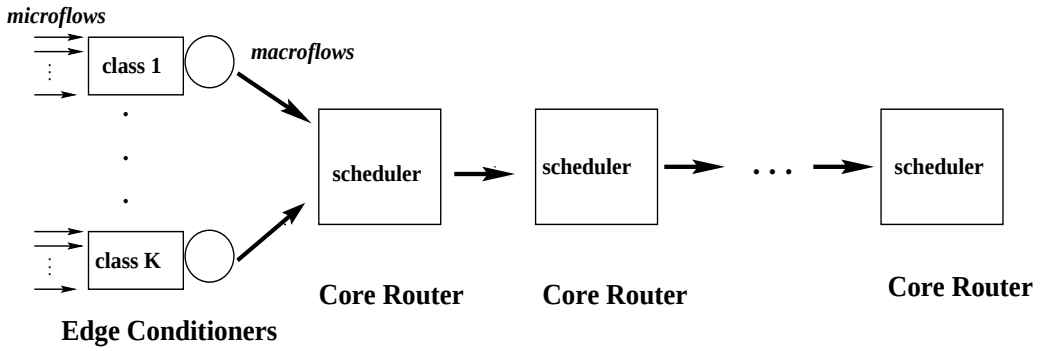


Figure 6.4: Class-based guaranteed services: dynamic flow aggregation along a path.

6.4 Admission Control with Dynamic Flow Aggregation

Traffic aggregation is a powerful technique that can be used to significantly reduce the complexity of both the data plane and the control plane of a network domain. This reduction in complexity *may* come at a price— that guaranteed services may only be provided to individual flows at a *coarser granularity*. In this section we address the problem of admission control for class-based guarantee services, where a fixed number of guaranteed delay service classes are offered in a network domain. The class-based guaranteed delay service model is schematically shown in Figure 6.4. A new user flow will be placed in one of the delay service classes if it can be admitted into the network. All flows in the same delay service class that traverse the same path will be aggregated into a single *macroflow*. This macroflow is shaped using an aggregate reserved rate at the edge conditioner, and is guaranteed with an end-to-end delay bound determined by the service class. We refer to the individual user flows constituting a macroflow as the *microflows*.

A key issue in the design of admission control for this class-based service model is the problem of *dynamic* flow aggregation. The dynamics come from the fact that *microflows may join or leave a macroflow at any time*. Hence the aggregate traffic profile for the macroflow may change dynamically, and as a result, the reserved rate for the macroflow may need to be adjusted accordingly. *This dynamic change in the aggregate traffic profile can cause some undesirable effect on the end-to-end delay experienced by the macroflow* (see Section 6.4.1). As far as we are aware, this problem of dynamic flow aggregation has *not* been identified nor addressed before in the literature. The existing work on traffic aggregation (in particular, in the context of guaranteed services, see, e.g. [40]) has implicitly assumed *static* flow aggregation: a macroflow is an aggregation of n *fixed* microflows, with no new microflows joining or existing constituent microflows leaving in the duration of

the macroflow. The remainder of this section is organized as follows. In Section 6.4.1 we first illustrate the impact of dynamic flow aggregation on providing end-to-end guaranteed services, and then in Section 6.4.2 we propose solutions to circumvent the problems using our BB architecture. In Section 6.4.3, we will briefly describe how to perform admission control for class-based guaranteed services.

6.4.1 Impact of Dynamic Flow Aggregation on End-to-End Delay

Before we illustrate the impact of dynamic flow aggregation, we first introduce some notation and assumptions. Consider a macroflow α which *currently* consists of n microflows. Let $(\sigma^j, \rho^j, P^j, L^{j,max})$ be the traffic profile of the microflow j, $1 \leq j \leq n$. For simplicity, we will use a dual-token bucket regulator, $(\sigma^\alpha, \rho^\alpha, P^\alpha, L^{\alpha,max})$, as the aggregate traffic profile for the macroflow α. Hence we have $\sigma^\alpha = \sum_{j=1}^{n} \sigma^j$, $\rho^\alpha = \sum_{j=1}^{n} \rho^j$, $P^\alpha = \sum_{j=1}^{n} P^j$, and $L^{\alpha,max} = \sum_{j=1}^{n} L^{j,max}$. Note that $L^{\alpha,max} = \sum_{j=1}^{n} L^{j,max}$ is because a packet of the maximum size may arrive from each of the n microflows at the same time. Hence the edge conditioner may see a burst of $L^{\alpha,max}$ at any time. In contrast, since only one packet from the macroflow α may leave the edge conditioner at any given time, the "maximum burst" the macroflow may carry into the network core is $\max_{j=1}^{n} L^{j,max}$. Let $\mathcal{P}$ denote the path of the macroflow α, and $L^{\mathcal{P},max}$ denote the maximum packet size permissible in a macroflow (i.e., a delay service class) along $\mathcal{P}$. Then $L^{\mathcal{P},max} \geq \max_{j=1}^{n} L^{j,max}$. Without loss of generality, we assume that $L^{\mathcal{P},max}$ is fixed.

Suppose we treat the macroflow α as *static*, i.e., with no microflows joining or leaving at any time. Let $\langle r^\alpha, d^\alpha \rangle$ be the rate-delay parameter pair reserved for the macroflow. For simplicity, assume that path $\mathcal{P}$ consists of only rate-based schedulers (h of them in total). Then from the end-to-end delay formula (6.3), the end-to-end delay experienced by the macroflow α (and therefore by any packets from any constituent microflows) is bounded by

$$d^\alpha_{end-to-end} = d^\alpha_{edge} + d^\alpha_{core} = T^\alpha_{on} \frac{P^\alpha - r^\alpha}{r^\alpha} + \frac{L^{\alpha,max}}{r^\alpha} + h \frac{L^{\mathcal{P},max}}{r^\alpha} + D^{\mathcal{P}}_{tot} \qquad (6.15)$$

where $D^{\mathcal{P}}_{tot} = \sum_{i \in \mathcal{P}} \Psi_i + \sum_{i \in \mathcal{P}, i \neq h} \pi_i$.

We claim that if we allow microflows to dynamically join or leave a macroflow, the end-to-end delay bound (6.15) may *no longer* hold. We illustrate this through an example. Consider a new microflow ν joins the existing macroflow α at time t^*. Let $(\sigma^\nu, \rho^\nu, P^\nu, L^{\nu,max})$

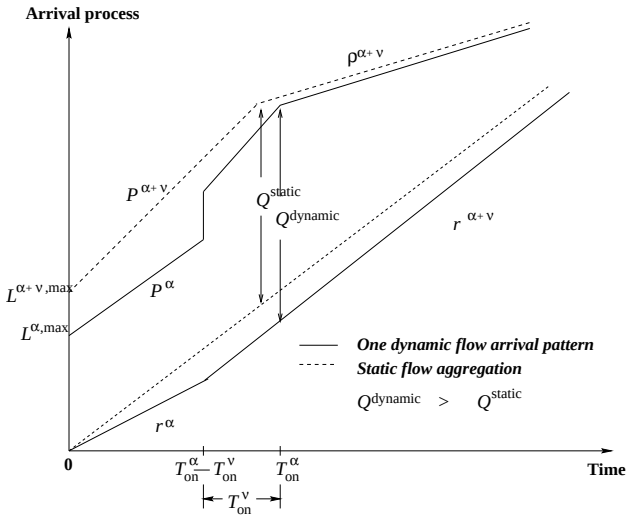


Figure 6.5: An example illustrating the edge delay bound violation when a new microflow joins.

be the traffic profile of the new microflow. Denote the "new" macroflow after the microflow ν has been aggregated (i.e., the macroflow that enters the network core after t^*) by α', and let $(\sigma^{\alpha'}, \rho^{\alpha'}, P^{\alpha'}, L^{\alpha',max})$ be its traffic profile. Suppose that the reserved rate for the "new" macroflow increases from r^α to $r^{\alpha'}$ at time t^*.

We first show that the packets from the "new" macroflow may experience a worst-case delay at the edge conditioner that is larger than $d^{\alpha'}_{edge} = T^{\alpha'}_{on}\frac{P^{\alpha'}-r^{\alpha'}}{r^{\alpha'}} + \frac{L^{\alpha',max}}{r^{\alpha'}}$. This can happen, for example, in the scenario shown in Figure 6.5. In this scenario, $T^\alpha_{on} \geq T^\nu_{on}$, and thus $T^\nu_{on} \leq T^{\alpha'}_{on} \leq T^\alpha_{on}$. We assume that all the constituent microflows of the existing macroflow α start at the same time (i.e., time 0) and are *greedy*: they dump the maximum allowed burst into the network at any time t, i.e., $A^\alpha(0,t) = \mathcal{E}^\alpha(t) = \min\{P^\alpha t + L^{\alpha,max}, \rho^\alpha t + \sigma^\alpha\}$. The new microflow ν joins the existing macroflow α at time $t^* = T^\alpha_{on} - T^\nu_{on}$, and it is also *greedy*: at any time $t \geq t^*$, $A^\nu(t^*,t) = \mathcal{E}^\nu(t-t^*) = \min\{P^\nu(t-t^*) + L^{\nu,max}, \rho^\nu(t-t^*) + \sigma^\nu\}$. Then it is not difficult to see that at time $t = T^\alpha_{on}$, the total amount of traffic that is queued at the edge conditioner is given by

$$Q(t) = (P^\alpha - r^\alpha)T^\alpha_{on} + (P^\nu + r^\alpha - r^{\alpha'})T^\nu_{on} + L^{\alpha',max}.$$

Hence the delay experienced by a packet arriving the edge conditioner at time $t = T^\alpha_{on}$ will be at least $Q(t)/r^{\alpha'}$, which can be shown to be larger than $d^{\alpha'}_{edge}$ in general. This larger delay is caused by the fact that at the time a new microflow is aggregated into an existing macroflow flow, the buffer at the edge conditioner *may not be empty*. The "old" packets

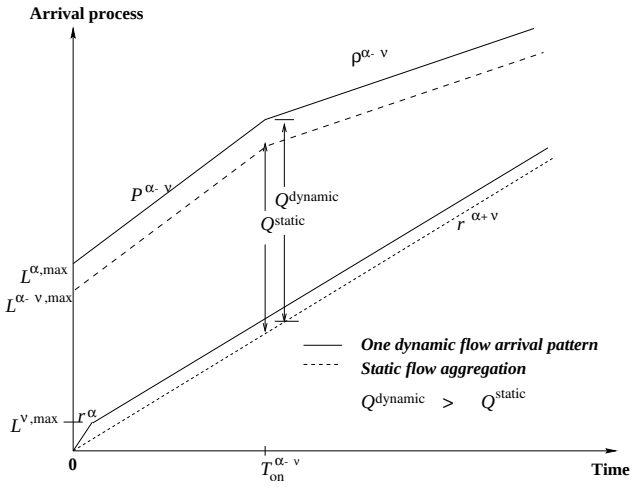


Figure 6.6: An example illustrating the edge delay bound violation when a constituent microflow leaves.

queued there can cause the "new" packets to experience additional delay that is no longer bounded by $d_{edge}^{\alpha'}$.

We now consider the delay experienced by packets from the "new" macroflow α' inside the network core. Despite the fact that packets from the "new" macroflow α' are serviced with a higher reserved rate $r^{\alpha'}$ ($\geq r^{\alpha}$), it can be formally established that some of these packets may experience a worst-case delay in the network core that is bounded by $d_{core}^{\alpha} = hL^{\mathcal{P},max}/r^{\alpha} + D_{tot}^{\mathcal{P}}$, not by $d_{core}^{\alpha'} = hL^{\mathcal{P},max}/r^{\alpha'} + D_{tot}^{\mathcal{P}}$. Intuitively, this can happen because the packets from the "new" macroflow may catch up with the last packets from the "old" macroflow. Hence they may be queued behind the "old" packets, incurring a worst-case delay bounded by d_{core}^{α} instead of $d_{core}^{\alpha'}$. Considering both the delay at the edge conditioner and that in the network core, we see that packets of the "new" macroflow may experience an end-to-end delay that is no longer bounded by the end-to-end delay formula (6.15).

A similar situation may also occur when a constituent microflow leaves an existing macroflow, *if we immediately decrease the reserved rate r^{α} to a new lower $r^{\alpha'}$*. For example, consider the scenario illustrated in Figure 6.6. Assume that a microflow ν dumps one packet with the maximum packet size of the microflow at time 0, and this packet is serviced first by the edge conditioner. Furthermore, we assume that the microflow leaves the system at the time $L^{\nu,max}/r^{\alpha}$. Suppose all other microflows in the macroflow are all greedy from time 0. Then it is not hard to see, if the reserved rate for α' is immediately reduced to $r^{\alpha'}$, that at

time $t = T_{on}^{\alpha'}$ the total amount of traffic that is queued at the edge conditioner is given by,

$$Q(t) = L^{\alpha',max} + T_{on}^{\alpha'} P^{\alpha'} - r^{\alpha'} T_{on}^{\alpha'} + L^{\nu,max} \frac{r^{\alpha'}}{r^{\alpha}}$$

Hence the delay experienced by a packet arriving the edge conditioner at time $t = T_{on}^{\alpha'}$ will be at least $Q(t)/r^{\alpha'}$, which is larger than $d_{edge}^{\alpha'}$.

In conclusion, we see that when a new microflow joins or a constituent microflow leaves an existing macroflow, the end-to-end delay experienced by the resulting macroflow after the microflow is aggregated or de-aggregated may not be bounded by the end-to-end delay formula (6.15). In other words, we cannot simply treat the resulting macroflow *as if it were a completely new and independent flow.* This is because when a new microflow joins or a constituent microflow leaves an existing macroflow, the buffer at the edge conditioner may not be empty in general. These packets queued at the edge conditioner may have a lingering effect that invalidates the end-to-end delay formula (6.15). *New apparatuses are thus needed to tackle the problem caused by dynamic flow aggregation.* Before we move on, we would like to comment that the problem of dynamic flow aggregation is *not* unique to the virtual time reference system used in this chapter. The same problem exists in a more general context. For example, dynamic flow aggregation will have the same effect on a network of WFQ schedulers, the reference system used in the IntServ model. This is because the situation happening at the edge conditioner described above will also apply to a WFQ scheduler.

6.4.2 End-to-End Delay Bounds under Dynamic Flow Aggregation

In this section we present new mechanisms to effectively circumvent the problems caused by dynamic flow aggregation. The basic objective of our approach is to enable the bandwidth broker to make admission control decisions at any given time, using only the traffic profile and reserved rate of the macroflow at that time. In other words, we do not want the bandwidth broker to maintain an elaborate history record of the microflow arrival and departure events of a macroflow. To take care of the potential extra delay at the edge conditioner, in Section 6.4.2.1 we introduce the notion of *contingency bandwidth* which is allocated for a short period of time (referred to as *contingency period*) after a microflow joins or leaves an existing macroflow to eliminate the lingering delay effect of the packets queued in the edge conditioner at the time the microflow joins or leaves an existing macroflow α. In Section 6.4.2.2 we extend the virtual time reference system to accommo-

date the problem of dynamic flow aggregation. With these new mechanisms implemented, we can show that the end-to-end delay of the macroflow after a microflow joins or leaves is bounded by a modified end-to-end delay formula. We shall see in Section 6.4.2.2 that this modified end-to-end delay bound can be computed based on the old delay bound (before a microflow joins/leaves) and the traffic profile of the "new" macroflow.

6.4.2.1 Contingency Bandwidth and Edge Delay Bound

We introduce the notion of *contingency bandwidth* to eliminate the lingering delay effect of the backlog queued in the edge conditioner at the time a microflow is aggregated into or de-aggregated from an existing macroflow. It works as follows: Suppose at time t^* a microflow ν joins or leaves an existing macroflow α. In other words, the traffic from the microflow will start arriving or stop arriving at the edge conditioner after t^*. Besides the reserved rate r^α being adjusted to a new reserved rate $r^{\alpha'}$ at t^*, *a contingency bandwidth Δr^ν is also temporarily allocated to the resulting "new" macroflow α' for a contingency period of τ^ν time units*. In other words, in addition to $r^{\alpha'}$, an extra Δr^ν amount of bandwidth is allocated to the "new" macroflow α' from time t^* to $t^* + \tau^\nu$, but de-allocated after time $t^* + \tau^\nu$. The contingency bandwidth Δr^ν and contingency period τ^ν is chosen in such a manner that the maximum delay in the edge conditioner experienced by any packet from the "new" macroflow α' after time t^* is bounded above by

$$d_{edge}^{new} \leq \max\{d_{edge}^{old}, d_{edge}^{\alpha'}\}. \tag{6.16}$$

where d_{edge}^{old} denotes the maximum edge delay bound on the "old" macroflow (i.e., before t^*) and $d_{edge}^{\alpha'} = T_{on}^{\alpha'}(P^{\alpha'} - r^{\alpha'})/r^{\alpha'} + L^{\alpha',max}/r^{\alpha'}$.

The following two theorems state the sufficient conditions on Δr^ν and τ^ν so that (6.16) holds, the proofs of which are fairly straightforward, which are delegated to Appendix C.1.

Theorem 14 (Microflow Join) *Suppose at time t^* a new microflow ν with the traffic profile $(\sigma^\nu, \rho^\nu, P^\nu, L^{\nu,max})$ joins an existing macroflow α. Let $r^\nu = r^{\alpha'} - r^\alpha$ and $Q(t^*)$ be the size of the backlog in the edge conditioner at time t^*. Then (6.16) holds if*

$$\Delta r^\nu \geq P^\nu - r^\nu \text{ and } \tau^\nu \geq \frac{Q(t^*)}{\Delta r^\nu}. \tag{6.17}$$

■

Theorem 15 (Microflow Leave) *Suppose at time t^* a constituent microflow ν with the traffic profile $(\sigma^\nu, \rho^\nu, P^\nu, L^{\nu,max})$ leaves an existing macroflow α. Let $r^\nu = r^\alpha - r^{\alpha'}$ and $Q(t^*)$ be the size of the backlog in the edge conditioner at time t^*. Then (6.16) holds if*

$$\Delta r^\nu \geq r^\nu \text{ and } \tau^\nu \geq \frac{Q(t^*)}{\Delta r^\nu}. \tag{6.18}$$

■

When a microflow ν joins or leaves an existing macroflow α, the BB can choose a contingency bandwidth allocation Δr^ν using the two theorems above. For example, when a microflow ν joins, we can set $\Delta r^\nu = P^\nu - r^\nu = P^\nu + r^\alpha - r^{\alpha'}$. Whereas when a microflow ν leaves, we can set $\Delta r^\nu = r^\nu = r^\alpha - r^{\alpha'}$. To compute the contingency period τ^ν *precisely*, we need to know the backlog $Q(t^*)$ in the edge conditioner at time t^*. Since at time t^* the maximum delay at the edge conditioner is bounded by d_{edge}^{old}, we have

$$Q(t^*) \leq d_{edge}^{old} r(t^*) = d_{edge}^{old}(r^\alpha + \Delta r^\alpha(t^*)) \tag{6.19}$$

where $r(t^*)$ is total bandwidth allocated to the macroflow at time t^*, which includes the reserved rate r^α and the *total* contingency bandwidth $\Delta r^\alpha(t^*)$ allocated to the macroflow α at time t^*. Given this upper bound on $Q(t^*)$, the BB can determine an upper bound $\hat{\tau}^\nu$ on the contingency period τ^ν as follows:

$$\hat{\tau}^\nu = d_{edge}^{old} \frac{r^\alpha + \Delta r^\alpha(t^*)}{\Delta r^\nu}. \tag{6.20}$$

Hence after $\hat{\tau}^\nu$, the BB can de-allocate the contingency bandwidth Δr^ν at time $t^* + \hat{\tau}^\nu$. We refer to this method of determining contingency period τ^ν as the (theoretical) *contingency period bounding* approach. This scheme does not require any feedback information from the edge conditioner regarding the status of its buffer occupancy. However in general it can be quite *conservative*.

A more *practical* approach is to have the edge conditioner to *feedback the actual contingency period* to the BB. This scheme is referred as the *contingency feedback* method, and works as follows. When a new microflow ν joins or leaves an existing macroflow α, the BB sends an *edge conditioner reconfiguration* message to the edge conditioner. In this message, in addition to the new reserved rate $r^{\alpha'}$, the contingency bandwidth Δr^ν is also

included. Suppose the edge conditioner receives this message at time t^*. It checks the current queue length $Q(t^*)$, and computes the new contingency period τ^ν. It can immediately inform the BB of the actual value of τ^ν. Or it can wait until $t^* + \tau^\nu$, and then send a *contingency bandwidth reset* message back to the BB. Note that whenever the buffer in the edge conditioner becomes empty, a *contingency bandwidth reset* message can also be sent back to the BB, resetting *all* of the contingency bandwidth allocated to the macroflow α (i.e., setting $\Delta r^\alpha = 0$). This is because after this point, the maximum delay experienced by any packets of the macroflow α is bounded by d^α_{edge}, which is solely determined by the *current* aggregate traffic profile of the macroflow α.

6.4.2.2 Extension to VTRS and Core Delay Bound

Recall that in the core stateless virtual time reference system, packets from the same flow carry virtual time stamps. These virtual time stamps satisfy, among the others, the *virtual time spacing* property, which depends on the reserved rate of the flow. The virtual time reference system is developed based on the assumption that the reserved rate of a flow is fixed. In this section we illustrate how the virtual time reference system can be extended to accommodate flow aggregation with dynamic rate changes. Based on this extension, we also derive a modified core-delay bound for flow aggregation.

Consider an existing macroflow α which traverses the path $\mathcal{P}$. For simplicity of exposition, we first assume that all the schedulers $\mathcal{S}_i$'s along the path are rate-based. Suppose that at time τ^*, the reserved rate of the macroflow α is adjusted at the edge shaper from r to r' (this happens every time the rate of the edge conditioner is adjusted). Let p^{k^*} be the last packet that leaves the edge conditioner before the rate change at τ^*, and p^{k^*+1} be the first packet that leaves the edge conditioner after the rate change at τ^*. Then for $k < k^*$, $\hat{a}_1^{k+1} - \hat{a}_1^k \geq L^{k+1}/r$, and for $k \geq k^*$, $\hat{a}_1^{k+1} - \hat{a}_1^k \geq L^{k+1}/r'$, where recall that $\hat{a}_1^k$ denotes the time packet p^k departs the edge conditioner and arrives at the first-hop scheduler.

To accommodate reserved rate change in the virtual time reference system, we need to modify the definition of the virtual time adjustment for the transitional packet p^{k^*+1} as follows. Define

$$\Delta^{k^*+1} = \max\left\{0, \Delta^{k^*} + h(\frac{L^{k^*}}{r} - \frac{L^{k^*+1}}{r'}) + \hat{a}_1^{k^*} - \hat{a}_1^{k^*+1} + \frac{L^{k^*+1}}{r'}\right\}. \tag{6.21}$$

For the packets after p^{k^*+1}, the definition of Δ^k is not changed, namely, it is given in (4.8) with r^j replaced by r'. Indeed we can show that for $k = k^* + 1, k^* + 2, \ldots$, Δ^k is the

cumulative delay experienced by packet p^k along the path $\mathcal{P}$ in the *ideal dedicated per-flow system* [86], where the rate of the servers is changed from r^α to $r^{\alpha'}$ at time τ^*.

With the above modification, we can show that the following theorem holds. For its proof, see Appendix C.2.

Theorem 16 *For $k = k^* + 1, k^* + 2, \ldots$, let $\delta^k = \Delta^k / h$, where for $k = k^* + 1$, Δ^k is given by (6.21), and for $k = k^* + 2, \ldots$, Δ^k is given in (4.8) with r replaced by r'. Then the virtual spacing and reality check properties hold for the macroflow after the rate change at τ^*. Namely, for $k = k^* + 1, \ldots$, $\tilde{\omega}_i^k - \tilde{\omega}_i^{k-1} \geq L^k / r'$, and $\hat{a}_i^k \leq \tilde{\omega}_i^k$, $i \in \mathcal{P}$. Furthermore, the delay experienced by these packets in the network core is bounded by the following modified core delay formula:*

$$\hat{f}_h^k - \hat{a}_1^k \leq h \max\left\{\frac{L^{\mathcal{P},max}}{r}, \frac{L^{\mathcal{P},max}}{r'}\right\} + D_{tot}^{\mathcal{P}}. \tag{6.22}$$

■

The above modified core delay bound is derived under the assumption that all the schedulers along path $\mathcal{P}$ are rate-based. We now consider the case where some schedulers along path $\mathcal{P}$ are *delay-based*. In order to ensure the validity of the virtual time reference system under this case, we need to impose an assumption[7]: *the delay parameter d^α associated with a macroflow α is fixed, no matter whether there are microflow arrivals or departures in the macroflow*. Under this assumption, delay-based schedulers can be easily incorporated into the extended virtual time reference system presented above. Suppose there are q rate-based schedulers and $h - q$ delay-based schedulers along path $\mathcal{P}$. Then the delay experienced by packets p^k's, $k = k^* + 1, k^* + 2, \ldots$ from the macroflow α after the rate change at τ^* is bounded by the following core delay formula:

$$\hat{f}_h^k - \hat{a}_1^k \leq q \max\left\{\frac{L^{\mathcal{P},max}}{r}, \frac{L^{\mathcal{P},max}}{r'}\right\} + (h - q)d^\alpha + D_{tot}^{\mathcal{P}}. \tag{6.23}$$

6.4.3 Admission Control with Dynamic Flow Aggregation

We now illustrate how to perform admission control and resource reservation with dynamic flow aggregation, based on the results obtained in Section 6.4.2.1 and Section 6.4.2.2. Consider a macroflow α. Let $D^{\alpha,req}$ be its end-to-end delay requirement, which we assume is

[7]This assumption is *not* too restrictive in a network where a number of *fixed* delay service classes are provided.

fixed throughout the *entire* duration of the macroflow. Whenever a microflow joins or leaves the macroflow α, we need to ensure that its end-to-end delay requirement is still satisfied. At a given time, let r^α be the *reserved* rate of macroflow α *excluding the contingency bandwidth allocated*. Let $\Delta r^\alpha(t)$ denote the *total* contingency bandwidth allocated to α at any time t. (Here we denote $\Delta r^\alpha(t)$ as a function to *emphasize its time dependent nature*, as every time a contingent period τ^ν expires, the corresponding contingency bandwidth is reduced from Δr^α.) Hence at any given time t the *actual* bandwidth allocated to the macroflow α is $r(t) = r^\alpha + \Delta r^\alpha(t) \geq r^\alpha$. Using this fact and (6.23), we see that if no new microflow joins or leaves, the delay in the network core experienced by packets of macroflow α is always bounded by $d^\alpha_{core} = qL^{\mathcal{P},max}/r^\alpha + (h-q)d^\alpha + D^{\mathcal{P}}_{tot}$, despite that the actual rate for macroflow α is not a constant. Let $\mathcal{P}$ be the path of macroflow α, and let $C^{\mathcal{P}}_{res}$ be the minimal residual bandwidth along path $\mathcal{P}$. Hence at most $C^{\mathcal{P}}_{res}$ *additional* amount of bandwidth (reserved and contingent) can be allocated to any macroflow along $\mathcal{P}$. We now consider how to deal with microflow joins and leaves. We consider these two cases separately below.

Microflow Join. Consider a new microflow ν wanting to join the existing macroflow α at time t^*. If the new microflow can be admitted, we need to determine, for the resulting "new" macroflow α', a new reserved rate $r^{\alpha'} \geq r^\alpha$ as well as Δr^ν amount of new contingency bandwidth for a contingency period of τ^ν. From Theorem 14, without loss of generality, we choose $\Delta r^\nu = P^\nu - r^{\alpha'} + r^\alpha$. In other words, during the contingency period, an additional P^ν amount of bandwidth is allocated to macroflow α. Hence in order to be able to admit the new microflow ν into the existing macroflow α, first of all, we must have $P^\nu \leq C^{\mathcal{P}}_{res}$. If this condition is satisfied, then we need to find the minimal new reserved rate $r^{\alpha'}$ so that the end-to-end delay requirement $D^{\alpha,req}$ can be satisfied for the resulting macroflow α'. Note that after contingency period, the edge queueing delay for any packets of the class is determined by the new class traffic profile and the reserved rate, therefore,

$$d^{\alpha'}_{end-to-end} = d^{\alpha'}_{edge} + \max\{d^\alpha_{core}, d^{\alpha'}_{core}\} \leq D^{\alpha,req}. \tag{6.24}$$

Since $r^{\alpha'} \geq r^\alpha$, $L^{\mathcal{P},max}/r^{\alpha'} \leq L^{\mathcal{P},max}/r^\alpha$. Hence $d^\alpha_{core} \leq d^{\alpha'}_{core}$. The constraint (6.24) is reduced to ensure that $d^{\alpha'}_{edge} \leq D^{\alpha,req} - d^\alpha_{core}$. From this constraint, $r^{\alpha'}$ can be easily computed. Hence the new microflow can be admitted if the new reserved rate $r^{\alpha'}$ can be accommodated along path $\mathcal{P}$ (i.e., if $\rho^\nu \leq r^{\alpha'} - r^\alpha \leq P^\nu \leq C^{\mathcal{P}}_{res}$).

If the microflow can be admitted, $r^\alpha + P^\nu$ is allocated to the macroflow during the contingency period (i.e., from t^* to $t^* + \tau^\nu$), and after $t^* + \tau^\nu$, only $r^{\alpha'}$ will be allocated for

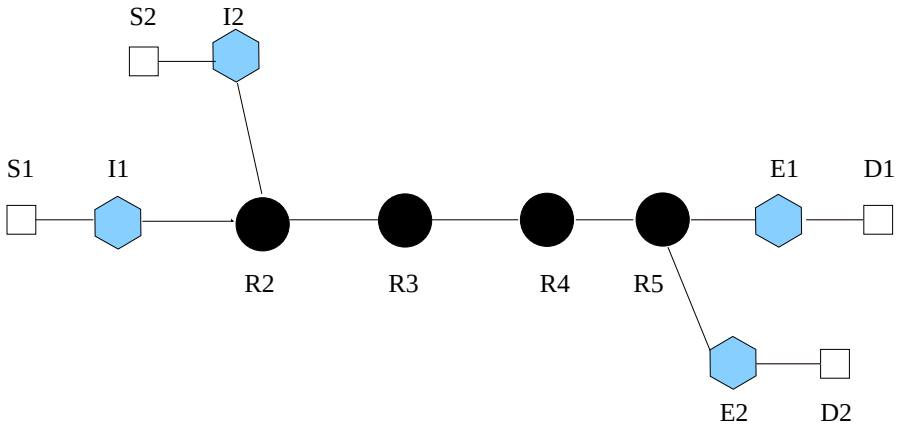


Figure 6.7: The network topology used in the simulations.

Table 6.1: Traffic profiles used in the simulations

Type	Burst size (b)	Mean rate (b/s)	Peak rate (b/s)	Max pkt size (B)	Delay Bounds (s)	
0	60000	0.05M	0.1M	1500	2.44	2.19
1	48000	0.04M	0.1M	1500	2.74	2.46
2	36000	0.03M	0.1M	1500	3.24	2.91
3	24000	0.02M	0.1M	1500	4.24	3.81

macroflow α'.

Microflow Leave. When a constituent flow ν leaves the macroflow α at time t^*, we may reduce the current reserved rate r^{α} to $r^{\alpha'}$. Clearly, we must ensure that the amount of bandwidth reduced, $r^{\nu} = r^{\alpha} - r^{\alpha'}$, is chosen such that the end-to-end delay requirement $D^{\alpha,req}$ must not be violated. Furthermore, *this reduction in reserved rate may not take place immediately in general.* From Theorem 15 and choosing $\Delta r^{\nu} = r^{\nu}$, we must continue to service the macroflow α with the current reserved rate r^{α} for a period of τ^{ν}. Only after the contingency period ends at $t^* + \tau^{\nu}$, can we reduce the current reserved rate by $r^{\nu} = r^{\alpha} - r^{\alpha'}$ amount. To determine $r^{\nu} = r^{\alpha} - r^{\alpha'}$, we must ensure that (6.24) holds. Since $r^{\alpha'} \leq r^{\alpha}$, we have $d^{\alpha}_{core} \leq d^{\alpha'}_{core}$. Hence we need to find a new reserved rate $r^{\alpha'}$ such that $d^{\alpha'}_{edge} + d^{\alpha'}_{core} \leq D^{\alpha,req}$. In either of these cases, the minimal $r^{\alpha'}$ (if it exists) that satisfies the end-to-end delay bound (6.24) can be found easily.

6.5 Simulation Investigation

In this section, we conduct simulations to explore the efficacy of our admission control algorithms for both per-flow and class-based guaranteed services. In particular, we compare the performance of our per-flow admission control algorithm with that used in IntServ

Guaranteed Service (GS) model. We also investigate the impact of dynamic flow aggregation on class-based guaranteed services.

Figure 6.7 depicts the network topology used in the simulations, where flows generated from source 1 (S1) are destined to destination 1 (D1) via the path connecting the ingress node (I1) to the egress node (E1), and flows generated from source 2 (S2) are destined to destination 2 (D2), via the path connecting the ingress node (I2) to the egress node (E2). Each ingress node consists of two components: edge conditioners; and a *core stateless* scheduler, which is the first-hop scheduler along the path. Let $x \rightarrow y$ denote the outgoing link from node x to node y. The capacity of outgoing links of all core routers is set to $1.5Mb/s$. The link capacity of $Si \rightarrow Ii$ and that of $Ei \rightarrow Di$, $i = 1, 2$, are assumed to be infinity. All the links are assumed to have zero propagation delay. We consider two simulation settings. In the first setting (*rate-based schedulers only*), all core routers employ $C_{\not S}$VC schedulers. In the second setting (*mixed rate/delay based schedulers*), schedulers employed for the outgoing links $I1 \rightarrow R2$, $I2 \rightarrow R2$, $R2 \rightarrow R3$, $R5 \rightarrow E1$ are $C_{\not S}$VCs, while those for $R3 \rightarrow R4$, $R4 \rightarrow R5$, and $R5 \rightarrow E2$ are VT-EDFs. The flow traffic profiles and possible delay requirements used in the simulations are listed in Table 6.1.

We first conduct a set of simulations to compare the efficacy of the admission control schemes (both per-flow and class-based) in the BB/VTRS model with the standard admission control scheme [34, 69] used for the GS in the IntServ model. In the GS model, the counterpart of a $C_{\not S}$VC scheduler is VC, while for VT-EDF, it is RC-EDF. The RC-EDF [34, 82] scheduler employs a per-flow shaper to enforce that the traffic of each flow entering the EDF scheduler conforms to its traffic profile. In this set of simulations, traffic is sent *only* from source $S1$ to destination $D1$ (i.e., there is no cross traffic). All flows are of type 0, and have the same end-to-end delay requirement (either $2.44\,s$ or $2.19\,s$). Moreover, each flow has an infinite lifetime. Note that under the per-flow guaranteed services, when the delay requirement of a type 0 flow is $2.44\,s$, a reserved rate equal to its mean sending rate will meet the delay requirement. Whereas, when the delay requirement is $2.19\,s$, a higher reserved rate is needed to meet the delay requirement. In the BB/VTRS aggregate scheme, a single delay service class is used, where the end-to-end delay requirement of the class is set to either either $2.44\,s$ or $2.19\,s$. For each flow in the class, a fixed delay parameter (cd) is used at all of the delay-based schedulers (this parameter will only be used in the mixed rate/delay-based scheduler setting). Simulations are conducted using three different values of cd ($0.10\,s$, $0.24\,s$ and $0.50\,s$). The objective of our simulation investigation is to compare the *maximum* number of flows that can be admitted under the three different

Table 6.2: Comparison of IntServ/GS, per-flow BB/VTRS and aggregate BB/VTRS schemes.

		Number of Calls admitted			
		Rate-Based Only		Mixed Rate/Delay-Based	
Delay bounds (s)		2.44	2.19	2.44	2.19
IntServ/GS		30	27	30	27
Per-flow BB/VTRS		30	27	30	27
Aggr BB/VTRS	cd = 0.10 (s)	29	29	29	29
	cd = 0.24 (s)			29	29
	cd = 0.50 (s)			29	28

admission control schemes: IntServ/GS, Per-flow BB/VTRS and Aggr BB/VTRS.

The simulation results are shown in Table 6.2. From the table we see that the IntServ/GS and Per-flow BB/VTRS schemes accept exactly the same number of flows under all the simulation settings. Whereas the Aggr BB/VTRS scheme has either slightly worse or better performance, depending on the end-to-end delay requirements of the flows. When the delay requirement is $2.44\,s$, the Aggr BB/VTRS scheme accepts one fewer flow than that can be accepted by either the IntServ/GS or Per-flow BB/VTRS scheme. This performance loss is due to contingency bandwidth allocation in the Aggr BB/VTRS scheme: when a new flow is accepted into the delay service class, an amount of bandwidth equal to its peak rate is reserved during the contigency period to avoid potential delay bound violation. In contrast, in both the IntServ/GS and Per-flow BB/VTRS schemes, the bandwidth reserved for the new flow is equal to its mean rate. However, when the delay requirement is $2.19\,s$, the Aggr BB/VTRS scheme can accept one or two more flows than that can be accepted by either the IntServ/GS or Per-flow BB/VTRS scheme. This performance gain is due to a number of factors: 1) each flow has precisely the same delay requirement as is provided by the delay service class; 2) the aggregate flow has a smaller core-delay bound than that of each individual flow in the per-flow guaranteed services; and 3) all flows have infinite life time, which, in this case, masks the *transient* effect of contingency bandwidth allocation used in the Aggr BB/VTRS scheme.

To better understand why the Aggr BB/VTRS scheme yields better performance in the case when the end-to-end delay requirement of the flows is $2.19\,s$, we examine more closely the bandwidth allocation allocated under the three schemes. Figure 6.8 plots the average bandwidth allocated to each flow using the three schemes (under the mixed rate/delay-based scheduler setting) as a function of the number of flows accepted into the network.

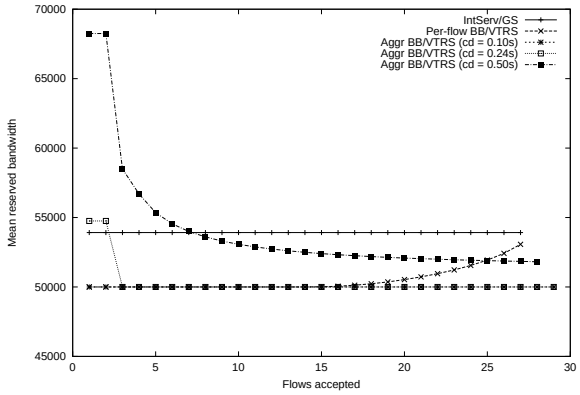


Figure 6.8: Mean reserved bandwidth.

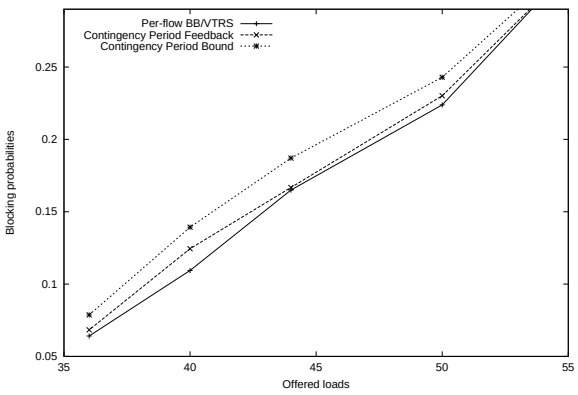


Figure 6.9: Flow blocking rates.

From the figure we see that under the Aggr BB scheme, the average reserved bandwidth per flow decreases, as more flows are aggregated into the delay service class. (Note in particular that with the fixed delay parameter $cd = 0.10\,s$, a per-flow bandwidth allocation that is equal to the mean rate of the flows is sufficient to support the end-to-end delay bound $2.19\,s$ of the delay service class.) The average reserved bandwidth eventually drops considerably below those of the Per-flow BB/VTRS and IntServ/GS schemes. As a result, under the Aggr BB/VTRS scheme there is sufficient residual bandwidth left to admit one or two more flows into the network. Under the Per-flow BB/VTRS scheme, a VT-EDF scheduler starts with allocating the minimum possible delay parameter to a flow, thereby producing the minimum bandwidth allocation (i.e., the mean rate of the flow). However, as more flows are admitted, the feasible delay parameter that can be allocated to a new flow becomes larger, resulting in higher reserved rate. As a result, the average reserved bandwidth per flow increases. It is interesting to note that although the Per-flow BB/VTRS and IntServ/GS admit the same number of flows (i.e., 27), the Per-flow BB/VTRS scheme has a slight smaller average reserved rate per-flow. Hence there is more residual bandwidth left under the Per-flow BB/VTRS scheme than that under the IntServ/GS scheme, albeit this residual bandwidth is not enough to admit another flow. This slight gain in the residual bandwidth is due to the ability of the Per-flow BB/VTRS scheme to perform path-wide optimization when determining the minimum feasible rate-delay parameter pair for a flow. In contrast, in the IntServ/GS scheme, the reserved rate of a flow is determined using the WFQ reference model, which then limits the range that the delay parameter can be assigned to the flow in an RC-EDF scheduler.

In the above simulations, we have assumed that all flows have infinite life time. We now conduct another set of simulations in which flows have finite holding times, and investigate

the impact of dynamic flow aggregation on the flow blocking performance of class-based guaranteed services. In this set of simuatations, flow holding time is generated using an exponential distribution with a mean of 200 seconds. Flows may originate from either of the two sources $S1$ or $S2$. We vary the flow inter-arrival times to produce various offered loads. We implement two versions of the aggregate BB/VTRS scheme: one using the contingency period bounding method, and another using the contingency period feedback method, as described in Section 6.4.2.1. Figure 6.9 shows the flow blocking rates of these two schemes as well as that of the per-flow BB/VTRS scheme, as we increase the flow arrival rates (and thus the offered load to the network). Each point in the plots of this figure is the average of 5 simulation runs. From the figure we can see that with dynamic flow arrivals and departures, the per-flow BB/VTRS scheme has the lowest flow blocking rate, as is expected. The theoretical contingency period bounding method has the worst flow blocking rate, because it uses the worst-case bound on the backlog of the edge conditioners. This leads to a portion of the link bandwidth used as the contingency bandwidth, which is not immediately released. Using the contingency period feedback method, the contingency period τ^{ν} is in general very small, thus the contingency bandwidth allocated is de-allocated in a very short period of time. In general, because it requires peak rate allocation at the time a new microflow arrives, the Aggr BB/VTRS schemes have a higher flow blocking rate than that of the per-flow BB/VTRS scheme. However, as the offered load increases, the flow blocking rates of these schemes converge. Hence as the network is close to its saturation point, the (transient) effect of contigency bandwidth allocation under the Aggr BB/VTRS scheme on the flow blocking performance becomes much less prominent.

6.6 Summary

In this chapter we have presented a novel bandwidth broker architecture for scalable support of guaranteed services that decouples the QoS control plane from the packet forwarding plane. More specifically, under this architecture, *core routers do not maintain any QoS reservation states, whether per-flow or aggregate*. Instead, the QoS reservation states are stored at and managed by a bandwidth broker. There are several advantages of such a bandwidth broker architecture. Among others, it avoids the problem of inconsistent QoS states faced by the conventional hop-by-hop, distributed admission control approach. Furthermore, it allows us to design efficient admission control algorithms without incurring any overhead at core routers. The proposed bandwidth broker architecture is designed based on a *core stateless* virtual time reference system developed in [86]. In this chapter we fo-

cused on the design of efficient admission control algorithms under the proposed bandwidth broker architecture. We consider both *per-flow* end-to-end guaranteed delay services and *class-based* guaranteed delay services with flow aggregation. Using our bandwidth broker architecture, we demonstrated how admission control can be done on an entire *path* basis, instead of on a "hop-by-hop" basis. Such an approach may significantly reduce the complexity of the admission control algorithms. In designing class-based admission control algorithms, we investigated the problem of dynamic flow aggregation in providing guaranteed delay services, and devised new mechanisms to effectively circumvent this problem. We conducted extensive analysis to provide theoretical underpinning for our schemes as well as to establish their correctness. Simulations were also performed to demonstrate the efficacy of our schemes.

Chapter 7

A Hierarchical Bandwidth Broker Architecture

7.1 Introduction

The objective of this chapter is to study the scaling issues in the centralized bandwidth broker model for flow-level dynamic QoS control and management. We consider the factors that may potentially affect the scalability of the centralized bandwidth broker model — in particular, we identify two major limiting factors: the memory and disk access speed and communication capacity between the bandwidth broker and edge routers. Because of the need to access and update the network QoS states during admission control operations, the number of memory and disk accesses/updates plays a dominant role in the time the bandwidth broker takes to process flow reservation requests. Therefore, reducing the overall number of QoS state accesses and updates is a key means to enhance the overall call processing capability of the bandwidth broker, thereby its scalability. In this chapter we develop a path-oriented, quota-based dynamic bandwidth allocation approach to address this issue. This approach is designed based on the two-level representation of the network QoS states proposed in [85], i.e., a path QoS state database representing the path-level QoS states as well as a link QoS state database representing the link-level QoS states of the network domain. By allocating bandwidth in units of quota to paths on demand, the proposed dynamic bandwidth allocation approach limits the majority of flow reservation requests to the path state accesses/updates only, avoiding the more time-consuming link state accesses and updates. As a result, the overall number of QoS state accesses and updates is significantly reduced, thus increasing the overall call processing capability of the centralized

bandwidth broker system.

This path-oriented, quota-based dynamic bandwidth allocation approach also leads to a natural architectural extension to the centralized bandwidth broker model: a hierarchically distributed bandwidth broker architecture to address the scaling problem caused by the potential communication bottleneck between the centralized bandwidth broker and edge routers. The proposed hierarchically distributed architecture consists of a number of edge bandwidth brokers, each of which manages a (mutually exclusive) subset of path QoS states and performs admission control for the corresponding paths, and a central bandwidth broker which maintains the link QoS state database and manages the quota allocation among the edge bandwidth brokers. We conduct extensive simulations to investigate the impact of the proposed mechanisms and architectural extensions on the network system performance, and to demonstrate their efficacy in enhancing the scalability of the centralized bandwidth broker model. Our study shows that the scalability issue of the centralized bandwidth broker model can be addressed effectively, without incurring any additional overhead at core routers.

The remainder of the chapter is organized as follows. In Section 7.2, we first present a centralized bandwidth broker architectural model, and then discuss the potential scaling issues of the centralized bandwidth broker architecture. In Section 7.3, we describe the basic path-oriented, quota-based dynamic bandwidth allocation approach, and study its efficacy in enhancing the overall call processing capability of the centralized bandwidth broker system. In Section 7.4 we present the hierarchically distributed multiple bandwidth broker architecture designed using the path-oriented, quota-based dynamic bandwidth allocation approach. Its impact on the system performance is investigated. We conclude the chapter in Section 7.6.

7.2 Bandwidth Broker Architecture: Basic Model and Scaling Issues

As the basis for our study, in this section we first present a basic centralized bandwidth broker architectural model and describe how admission control is performed under such a model. We then discuss the potential scaling issues in this centralized bandwidth broker model, and briefly outline the solutions that we will develop in this chapter to address these issues.

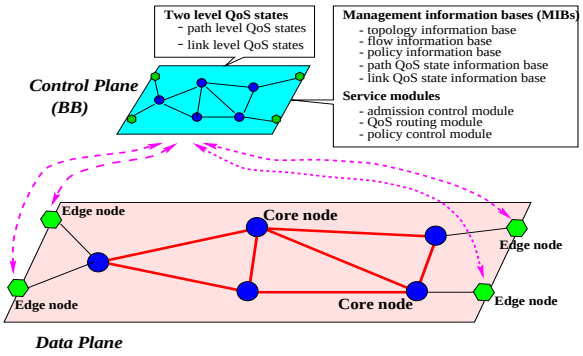


Figure 7.1: Illustration of a bandwidth broker.

7.2.1 The Basic Bandwidth Broker Model

The basic centralized bandwidth broker (BB) model for the management and control of the QoS provisioning of a network domain is schematically depicted in Figure 7.1. This model is based on the bandwidth broker architecture proposed in [85]. In this architectural model, the bandwidth broker centrally manages and maintains a number of management information (data)bases (MIBs) regarding the network domain. Among them, the network topology database and network QoS state databases are most relevant to the study of this chapter. The network topology database and network QoS state databases together provide a logical representation (i.e., a QoS abstraction) of the network domain and its entire state. With this QoS abstraction of the network domain, the bandwidth broker performs QoS control functions by managing and updating these databases. In this sense, the QoS control plane of the network domain is decoupled from its data plane. The core routers of the network domain are removed from the QoS control plane: core routers do not maintain any QoS reservation states, whether per-flow or aggregate, and do not perform any QoS control functions such as admission control.

In our centralized bandwidth broker model, the network QoS states are represented at two levels: link-level and path-level. The link QoS state database maintains information regarding the QoS state of each link in the network domain, such as the total reserved bandwidth or the available bandwidth of the link. The path QoS state database maintains the QoS state information regarding each path of the network domain, which is extracted and "summarized" from the link QoS states of the links of the path. An example of the path QoS state is the available bandwidth along a path, which is the minimal available bandwidth among all its links. As shown in [85], by maintaining a separate path-level QoS state, the bandwidth broker can conduct fast admissibility test for flows routed along the path. Furthermore,

path-wise resource optimization can also be performed based on the (summarized) path QoS state. As will be demonstrated in this chapter, *this two-level representation of the network QoS states is also the key feature that leads to scalable design of bandwidth broker architectures for dynamic flow-level QoS provisioning.* Lastly, we note that both the link QoS states and path QoS states are aggregate QoS states regarding the links and paths. No per-flow QoS states are maintained in either of the two QoS databases. The QoS and other control state information regarding each flow[1] such as its QoS requirement and reserved bandwidth is maintained in a separate flow information database managed by the bandwidth broker [85].

We now briefly describe a simple admission control scheme to illustrate how flow-level dynamic QoS provisioning can be performed under the basic centralized bandwidth broker model. For simplicity of exposition, throughout this chapter we assume that bandwidth is the critical network resource that we are concerned about. We consider the flow reservation set-up request first. When a new flow arrives at an edge router, requesting a certain amount of bandwidth to be reserved to satisfy its QoS requirement, the flow reservation set-up request is forwarded by the edge router to the bandwidth broker. The bandwidth broker then applies an admissibility test to determine whether the new flow can be admitted. More specifically, the bandwidth broker examines the path QoS state (obtained from the corresponding link states) and determines whether there is sufficient bandwidth available along the path to accommodate the new flow. If the flow can be admitted, the bandwidth broker updates the path QoS state database and link QoS state database (as well as the flow information database) to reflect the new bandwidth reservation along the path. If the admissibility test fails, the new flow reservation set-up request will be rejected, and no QoS information databases will be updated. In either case, the BB will signal the ingress edge router of its decision. For a flow reservation tear-down request, the bandwidth broker will simply update the corresponding link state database and path state database (as well as the flow information database) to reflect the departure of the flow. Clearly, using the basic admission control scheme presented above, processing either the flow reservation set-up or tear-down request requires access/update to the link QoS state database as well as the path QoS state database. Access and update of the link QoS states are necessary to ensure that the link QoS states are always up-to-date, so that the bandwidth broker can obtain accurate path QoS state information and make correct admission control decisions. We refer to this

[1] In this chapter a flow can be either an individual user flow, or an aggregate flow of multiple users, defined in whatever appropriate manner (e.g., an aggregate flow representing traffic from an institution or a sub-network).

"naive" admission control scheme that requires per-flow link QoS state access/update, as the *link-update* admission control scheme. In this chapter we will present a more efficient approach to performing bandwidth allocation and admission control that can significantly reduce the overall number of QoS state accesses and updates.

7.2.2 Scaling Issues

The issue of scalability is an important consideration in the design of a (centralized) bandwidth broker system. An important measure of scalability is the ability of the bandwidth broker system to handle large volumes of flow reservation requests, as the network system scales. For example, as the network link capacity increases, the *call processing capability* of the bandwidth broker system, defined as the number of flow requests that can be processed by the bandwidth broker system per unit of time, must scale with the increasing number of flows that can be accommodated in the network system. In particular, the bandwidth broker system should *not* become the bottleneck while the network system has not been overloaded.

Although it is possible to enhance the call processing capability of a bandwidth broker system by simply adding more processing power or increasing memory and disk access speed, such an approach *in itself* in general does not provide a scalable solution. To develop a scalable bandwidth broker architecture, we need to gain a fundamental understanding of the potential scaling issues and problems in a centralized bandwidth broker architecture, and then devise appropriate mechanisms and architectural extensions to address these issues and problems. This is precisely the objective of this chapter. In this section we will identify two key factors that can potentially limit the scalability of the centralized bandwidth broker architecture, and outline the solutions we will develop in the remainder of the chapter.

There are many factors that may potentially affect the call processing capability of a bandwidth broker system. Among them, the speed of memory and disk accesses plays a prominent role. Recall that when processing a flow reservation set-up request, the bandwidth broker must perform an admissibility test, and if the request can be granted, update the relevant QoS states. Likewise, when processing a flow reservation tear-down request, the bandwidth broker needs to update the relevant QoS states. In either case, access and/or update to QoS states are involved. Since memory/disk access speed is typically much slower than processing speed, we argue that the processing time of flow requests is determined in a large part by the number of memory/disk accesses and updates. Therefore, an important

means to enhance the call processing capability of a bandwidth broker system is to reduce the number of accesses and updates to the network QoS states maintained by the bandwidth broker.

Another factor that may affect the overall call processing capability of a centralized bandwidth broker system is the capacity of the communication channels (e.g., the network or I/O bandwidth) between a centralized bandwidth broker system and various edge routers. As the number of flows increases, these communication channels can become a potential bottleneck, limiting the number of flow requests delivered to the centralized bandwidth broker system, thereby reducing its overall call processing capability. To scale with the demand of the network system, a distributed multiple-bandwidth-broker architecture may be called for. Therefore, architectural extension to the basic centralized bandwidth broker model must be considered.

The focus of this chapter is on the design of mechanistic enhancement and architectural extension to the basic centralized bandwidth broker model to improve its scalability. In particular, we propose and develop a *path-oriented, quota-based* (in short, PoQ) dynamic bandwidth allocation mechanism for performing admission control. This PoQ dynamic bandwidth allocation mechanism exploits the two-level representation of the network QoS states used in the basic centralized bandwidth broker model, and attempts to avoid accessing and updating the link QoS states every time a flow reservation set-up or tear-down request is processed. In other words, this mechanism limits the majority of accesses and updates to path QoS states only, thereby reducing the overall number of accesses and updates to the network QoS states. The basic PoQ mechanism is described in detail in Section 7.3. Using the PoQ dynamic bandwidth allocation mechanism, in Section 7.4, we extend the basic centralized architecture with a single bandwidth broker to a hierarchically distributed architecture with multiple bandwidth brokers to address the scaling problem posed by the potential communication bottleneck between the bandwidth broker system and the edge routers. Our results demonstrate that the proposed path-oriented and quota-based dynamic bandwidth allocation mechanism is indeed an effective means to increase the overall call processing capability of the bandwidth broker. Furthermore, the bandwidth broker architecture can be designed in such a manner that it scales, as the network capacity increases.

15. op_p: if $op_p == 0$, path p is in the normal mode.
16. if $op_p > 0$, path p is in the critical mode.
17. cl_p: list of critical links along path p.
18. R_p: total reserved rate along path p.
19. Q_p: number of quotas allocated to path p; it also denotes
20. the total quota bandwidth along p, if no confusion.
21. aqb_p: available quota bandwidth on p: $aqb_p = Q_p - R_p$.
22. op_l: if $op_l == 0$, link l is not critical.
23. if $op_l == 1$, link l is critical.
24. C_l: capacity of link l.
25. Q_l: total quotas of link l.
26. aq_l: available quota of link l. $aq_l = Q_l - \sum_{p:l \in p} Q_p$.
27. rb_l: residual bandwidth of link l. $rb_l = C_l - \sum_{p:l \in p} R_p$.

Figure 7.2: Notation used in the algorithm.

7.3 Single Bandwidth Broker Design

In this section we present the path-oriented, quota-based (PoQ) mechanism for dynamic bandwidth allocation under a single bandwidth broker. We first describe the basic operation of the mechanism (the base scheme), and then analyze its complexity and performance. Simulation results are presented at the end of the section to illustrate the efficacy of the scheme.

7.3.1 The Basic PoQ Scheme

As pointed out in Section 7.2, using the basic link-update admission control scheme to process a flow reservation set-up or tear-down request, the bandwidth broker needs to access/update the link QoS state of each link along the flow's path. Hence the per-flow request processing time is proportional to the number of link state accesses/updates. As the volume of flow requests increases, these per-flow QoS state accesses/updates can slow down the operations of the bandwidth broker, limiting its flow processing capability. Therefore, reducing the number of link QoS state accesses/updates is an important means to prevent the bandwidth broker from becoming a potential bottleneck.

In this section we present the basic PoQ dynamic bandwidth allocation mechanism for a single centralized bandwidth broker, and illustrate how it can be employed to reduce the overall number of link QoS state accesses and updates. We first outline the basic ideas behind the PoQ mechanism, and then provide a more formal and detailed description. Under the basic PoQ scheme, the total bandwidth of each link of the network is (virtually) divided into *quotas*. A quota is a "chunk" of bandwidth, appropriately chosen, that is much larger than the average bandwidth requirement of typical flows. *Bandwidth is normally allocated*

```
0.   Upon an arrival of a new flow f at a path p:
1.     case 1: (op_p == 0 and aqb_p ≥ r_f)
2.       R_p ← R_p + r_f; accept the flow; return.
3.     case 2: (op_p == 0 and aqb_p < r_f)
4.       request more quota on all the links l: l ∈ p (Fig. 4);
5.     case 3: (op_p > 0)
6.       request bandwidth r_f on all critical links: l ∈ cl_p (Fig. 4);
7.       for l ∉ cl_p
8.         if (aqb_p < r_f) request more quota (Fig. 4);
9.     if (all requests are granted)
10.      update Q_p if more quotas are allocated;
11.      R_p ← R_p + r_f; accept the flow; return.
12.    else reject the flow reservation set-up request.
```

Figure 7.3: Path level admission control.

on-demand to each path in units of quotas. To be more precise, bandwidth allocation along a path operates in two possible modes: the *normal* mode and *critical* mode. During the normal mode, the bandwidth broker allocates and de-allocates bandwidth in unit of one quota at a time. The path QoS state of a path maintains the number of quotas of bandwidth that have been allocated to the path, in addition to the actual bandwidth that has been reserved for the flows routed along the path. When a flow reservation set-up request along a path arrives, the bandwidth broker only needs to check the corresponding path QoS state to see whether the quotas of bandwidth allocated to the path are sufficient to satisfy the flow's request. If the answer is positive, the flow request is accepted, and the relevant path QoS state is updated accordingly (i.e., the actual reserved bandwidth along the path is increased by the amount requested by the flow). Similarly, when a flow reservation tear-down request arrives, the bandwidth broker simply needs to update the relevant path QoS state (i.e., the actual reserved bandwidth of the path is decreased by the amount reserved for the flow). We see that in the above two cases, flow requests can be processed by accessing and updating the path QoS states only, without the need to access/update the link QoS state database. When there are a large number of flows arriving and departing in a short period of time, with an appropriately chosen quota size we expect that many of these flow requests (either reservation set-up or tear-down) will be processed by the bandwidth broker using only the path QoS states. This key observation is the major motivation behind our PoQ dynamic bandwidth allocation mechanism.

In the case that the bandwidth allocated to a path is not sufficient to satisfy the reservation set-up request of a flow, the bandwidth broker will attempt to allocate a new quota to the path to accommodate the flow reservation set-up request. In this case, the bandwidth broker needs to check each link QoS state along the path to see whether there is a quota available at all the links. If this is the case, a new quota is allocated to the path, and the number of

available quotas at each link of the path is decremented by 1. When there is an extra unused quota available along a path (due to flow departures), the extra quota will be re-claimed by the bandwidth broker, and the extra quota is returned to each link along the path. The available number of quotas at these links will be increased by 1. Clearly, quota allocation and de-allocation incur some overhead. In particular, the bandwidth broker needs to access and update the link QoS states to keep track of the available quotas at each link. Generally speaking, large quota size tends to reduce the overhead of quota management. On the other hand, large quota size has other performance implications, as we will see later.

Quota allocation for a path can fail if one of the links along the path does not have sufficient quotas left. In this case, bandwidth allocation for the path enters into the *critical* mode. More generally, when the available quota of a link falls below a threshold (say, no quota left), we say that the link is *critically loaded* (or the link is critical). When a link is critically loaded, all paths traversing this link enter the critical mode. Once a path is in the critical mode, the bandwidth broker will cease allocating bandwidth along the path in units of quota. Instead, the bandwidth is allocated or de-allocated on a per-flow basis, as in the basic link-update scheme described in Section 7.2. In particular, it maintains an accurate link QoS state for each critically loaded link (e.g., the precise amount of reserved bandwidth at the link). Hence when processing a flow reservation set-up or tear-down request for a path in the critical mode, the bandwidth broker must access and update the link QoS states of those critically loaded links along the path. In this way, we ensure that the admission control decision is always made correctly. We switch to the link-update admission control scheme so that no flow reservation set-up request will be rejected unnecessarily. As a result, whenever a flow is admitted using the link-update scheme, it will also be admitted using the basic PoQ scheme. In the next section, we will consider a "lossy-path" model in the context of the multiple bandwidth broker. The "lossy-path" model can also be used in combination with the basic PoQ scheme to reduce the link QoS state access/update overhead.

In the above we have provided an outline of the basic PoQ dynamic bandwidth allocation scheme. A more formal and detailed description of the scheme is presented in pseudo-code in Figures 7.3, 7.4, and 7.5. Figure 7.2 summarizes the notation used in the description. For the ease of exposition, the scheme is divided into three function blocks. Figure 7.3 describes the path-level admission control for flow reservation set-up and quota allocation management. Figure 7.4 describes the link-level bandwidth allocation and quota allocation management. Finally, Figure 7.5 describes both the path-level and link-level bandwidth and quota management operations for handling flow departures.

```
0.  Upon a path p requests r_p on a link l:
1.    /* r_p can be a quota or a flow's request rate */
2.    case 1: (op_l == 0 and aq_l < r_p)
3.      collect residual bandwidth: rb_l ← C_l − Σ_{p:l∈p} R_p;
4.      if (rb_l < r_p) reject the request; return.
5.    case 2: (op_l == 1 and rb_l < r_p) reject the request; return.
6.    /* The request can be honored */
7.    if (op_l == 0 and aq_l < r_p)
8.        op_l ← 1; /* transition: normal → critical */
9.      for (p' : l ∈ p')
10.       cl_{p'} ← cl_{p'} ∪ l; op_{p'} ← op_{p'} + 1;
11.   case 1: (op_l == 0) aq_l ← aq_l − 1
12.   case 2: (op_l == 1) rb_l ← rb_l − r_p.
```

Figure 7.4: Link level bandwidth/quota allocation.

```
0.  Upon an existing flow f departs on a path p:
1.    R_p ← R_p − r_f;
2.    if (op_p > 0)
3.      for (l ∈ cl_p)
4.        rb_l ← rb_l + r_f; recompute aq_l;
5.        if (aq_l ≥ 0) /* transition: critical → normal */
6.          for (p' : l ∈ p')
7.            op_{p'} ← op_{p'} − 1; set Q_{p'};
8.            cl_{p'} ← cl_{p'} − l;
9.    else if (op_p == 0 and p has excess quota)
10.     Q_p ← Q_p − 1; /* return excess quota */
11.     for (l ∈ p)
12.       aq_l ← aq_l + 1;
```

Figure 7.5: Scheme for handling flow departure.

7.3.2 Complexity and Performance

In this section, we will provide a simple analysis of the complexity of the basic PoQ dynamic bandwidth allocation scheme, and compare it with the link-update admission control scheme in terms of the QoS state access/update overhead. Since the path QoS states are always accessed/updated for every flow reservation set-up or tear-down request, we focus on the number of link QoS state accesses/updates. We measure the complexity of the PoQ scheme by the *expected cost of link QoS state access/update per flow*, i.e., the number of link QoS state accesses and updates incurred by processing a flow arrival and departure.

Consider a network domain whose average path length is P. Let ϕ be the probability that an "average" path of length P is in the critical mode, and γ be the probability a flow is rejected due to unavailability of bandwidth along the path. Note that under the basic PoQ scheme, a flow can only be rejected when the path is in the critical mode. In addition, let φ and χ denote, respectively, the probability that the flow reservation set-up request triggers a quota allocation, and the probability that the flow reservation tear-down request triggers a quota de-allocation, conditioned on that the flow is admitted. Then the expected link access/update cost, denoted by Θ_{PoQ}, is given by the following expression:

$$\Theta_{PoQ} = P\gamma + 3P\phi(1-\gamma) + P(\varphi+\chi)(1-\gamma). \tag{7.1}$$

The first term in the above expression is the number of link QoS state accesses for a flow that is rejected. The second term is the number of link QoS accesses and updates for processing a flow arrival in the critical mode plus the number of link QoS state updates for processing a flow departure in the critical mode. Here we assume that to admit a flow in the critical mode, the relevant link states are first accessed for the admissibility test, and then updated after the flow is admitted. Note also that for a flow admitted in the normal mode, no link QoS state is accessed or updated. The last term in (7.1) reflects the overhead of quota allocation and de-allocation.

Comparing the expected link QoS state access/update cost of the PoQ scheme with that of the naive link-update admission control scheme, $\Theta_{L-U} = P\gamma + 3P(1-\gamma)$, we see that the reduction in the per-flow link QoS access/update cost under the PoQ scheme is (approximately) proportional to $1-\phi$. Hence if the network system can accommodate N flows, then the reduction in the total link QoS access/update cost is in the order of $N(1-\phi)$. For a large N, this can amount to significant cost reduction, even when ϕ is fairly close to 1 (say, $\phi = 0.9$). On the other hand, this reduction in the link QoS state access/update

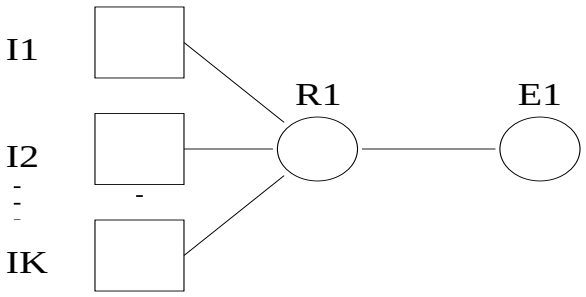


Figure 7.6: Topology used in the simulations.

cost is offset to some degree by the overhead of quota allocation and de-allocation. Hence judicious choice of quota size is important in controlling this overhead and balancing the overall cost reduction. This issue will be investigated in our simulation study reported in the next section.

Before we leave this section, we comment that the complexity of the PoQ scheme can be analyzed formally using queueing theory. In particular, under the assumption of exponential flow arrival and departure, the PoQ scheme can be modeled as a Markovian system, and the probabilities γ, ϕ, φ and χ can be derived either precisely or approximately. A key result from the analysis is that as the network capacity increases (thus the number of flows that can be accommodated also increases), the probability ϕ that a path enters the critical mode decreases, while the normalized network load (defined as the ratio of the offered load to the network capacity) is fixed. As we will see in the next section, this observation is also supported by our simulation results. Hence the PoQ scheme indeed improves the scalability of the centralized bandwidth broker model.

7.3.3 Simulation Investigation

In this section we conduct simulations to study the performance of the basic PoQ scheme. In particular, we will investigate the impact of quota size on the performance of the scheme and its scaling property as the network capacity increases.

Table 7.1: Call admission and quota allocations (C = 5400).

normalized load	a = 0.95					a = 1.00				
quota size	30	60	100	120	150	30	60	100	120	150
total flow arrivals	22946	22946	22946	22946	22946	24099	24099	24099	24099	24099
total accepted flows	22946	22946	22946	22946	22946	23878	23878	23878	23878	23878
flows accepted in normal	22946	22946	22570	22464	22395	17519	11204	7582	7370	7319
flows accepted in critical	0	0	376	482	551	6359	12674	16296	16508	16559
quotas allocated	736	396	220	155	39	499	114	6	0	0
quotas deallocated	739	397	222	156	39	499	114	6	0	0

Since using the PoQ scheme the QoS state of a link is only accessed and updated when the link becomes critical, in our simulations we use a simple network topology with a bottleneck link to study the cost of link QoS state accesses and updates. This simple topology allows us to focus on the key features of the PoQ scheme and provide an adequate environment to explore its performance. The network topology is shown in Figure 7.6, where K ingress routers, $I1, I2, \ldots, IK$, are connected via a core router $R1$ to an egress router $E1$. The link $R1 \rightarrow E1$ is the *bottleneck* link of the network topology, and the links $Ii \rightarrow R1$ are assumed to have infinite capacity, $i = 1, 2, \ldots, K$. Flows arriving at the ingress routers have an exponential interarrival time with its mean denoted by $1/\lambda$, and an exponential holding time with its mean denoted by $1/\mu$. In our simulations the mean flow holding time $1/\mu$ is fixed at 900 seconds, while we vary the mean flow interarrival time to produce different offered load. The *offered* load ρ is λ/μ, which represents the *average* number of flows that may exist in a system (if no flows are blocked). Each flow requests a unit of bandwidth, and the bottleneck link $R1 \rightarrow E1$ has C units of bandwidth. Hence the maximum number of flows that can be accommodated by the bottleneck link is C. We introduce the *normalized network load* $a = \rho/C$ as a metric for measuring how heavy the bottleneck link is loaded. For example, if $a < 1$, then the offered load (i.e., the average number of flows that may exist at any time) is less than what can be accommodated by the bottleneck link, the system is not overloaded. Otherwise, the network system is overloaded. In our simulation study, all simulations last 10000 simulated seconds, of which 6000 seconds are the warm-up times. Each value reported in the results is the mean value of 5 simulation runs with different random seeds for the mean flow interarrival times.

In the first set of simulations, we examine the impact of quota size on the performance of the scheme. In this set of simulations, the bottleneck link capacity C is 5400. The number of paths K is set to 3, i.e., we have three ingress routers, I1, I2, and I3. Flow arrivals are uniformly distributed onto the three ingress routers. We conduct simulations using five different quota sizes, namely 30, 60, 100, 120, and 150. The simulation results are summarized in Table 7.1 under two different normalized loads (a=0.95 and a=1.00). In this table, we list the total number of flow arrivals at all the ingress routers, the total number of flows accepted into the network system, and among the flows accepted, the total number of flows accepted in the normal mode as well as the total number of flows accepted in the critical mode. We also list the total number of quota allocation and de-allocation operations performed by the bandwidth broker after the warm-up period (i.e., when the network system is in a more stable state).

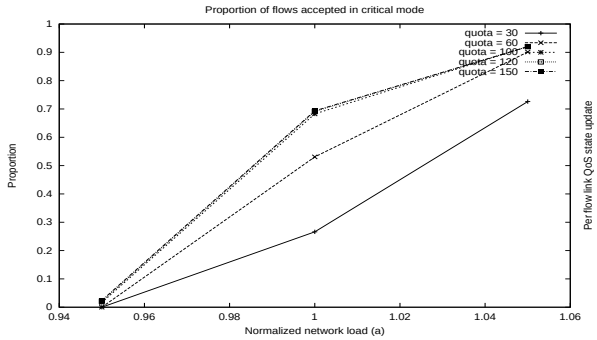


Figure 7.7: Proportion of flows accepted in critical mode (C = 5400).

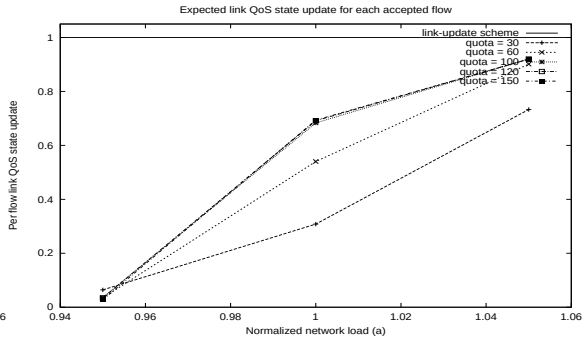


Figure 7.8: Expected link level QoS update/accepted flow (C = 5400).

From Table 7.1 we see that in the case of $a = 0.95$, i.e., the network is relatively light-loaded, the majority of the flows are accepted in the normal mode. In particular, when the quota sizes are 30 and 60, all flows are accepted in the normal mode, whereas when the quota size increases to 100, 120 and 150, only a few hundreds of flows are accepted in the critical mode. Hence in this light-load case, the portion of calls accepted in the critical mode is very small. In contrast, in the case of $a = 1.00$, i.e., the network is now heavily loaded, the portion of flows accepted in the critical mode increases significantly. In particular, when the quota sizes are large, the majority of flows are accepted in the critical mode. Figure 7.7 shows the portion of flows accepted in the critical mode with the five different quota sizes, as the normalized network load increases. We see that when the network is relatively light-loaded (say, $a \leq 0.95$), the quota size has little impact on the portion of the flows accepted in the critical mode. However, as the network load increases, the impact of the quota size is more significant. In particular, in the case that the quota size is 60 or bigger more than half of the flows are accepted in the critical mode when the normalized load reaches $a = 1.00$. Hence when the network is over-loaded, the quota size has a significant impact on the performance of the PoQ scheme. However, it is also interesting to observe that in the heavy load cases, increasing the quota size beyond a certain value (say, 100) does not seem to have any further impact.

We now shift our attention to the cost of the PoQ scheme, namely, the expected cost of link QoS state access/update. To simplify discussion, we focus on the link QoS state update cost incurred by flow arrivals and the overhead of quota allocation and de-allocation. Hence, instead of the expected cost of link QoS state access/update per flow, Θ_{PoQ}, defined in Section 7.3.2, we use a simplified metric, the expected cost of link QoS state update for

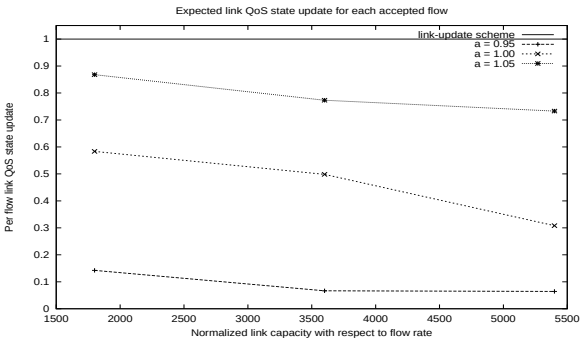
accepted flows, defined below:

$$\hat{\Theta}_{PoQ} = \frac{M + G + L}{N}, \tag{7.2}$$

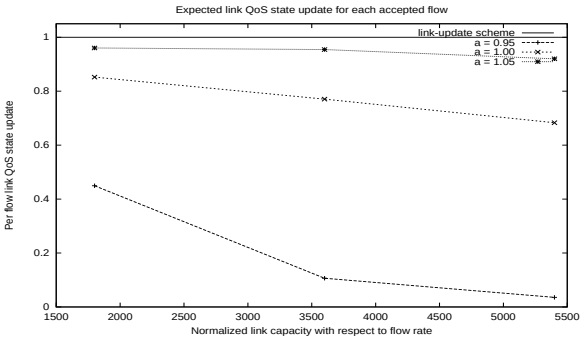
where N denotes the total number of accepted flows, M the number of flows accepted in the critical mode, and G and L denote a number of quota allocations and de-allocations, respectively.

Figure 7.8 shows the expected cost of link QoS state update per accepted flow as a function of the normalized network load for various quota sizes. The bottleneck link capacity C is set to 5400. From the figure we see that when the normalized network load is below 0.98, the expected cost of link QoS state update per accepted flow is less than 0.5 for all the quota sizes. Hence on the average more than half of the flows accepted do not require any link QoS updates. Even for the network is heavily overloaded, say, $a = 1.03$, the expected cost of link QoS state update per accepted flow is still less than 0.8. In other words, the PoQ scheme is capable of reducing the overhead of per-flow processing even at the heavily loaded scenarios. In general, smaller quota sizes tend to have better performance when the network is heavily loaded. This is because the link QoS update cost is dominated by the cost incurred by flows accepted in the critical mode. On the other hand, when the network is not heavily loaded (say, $a = 0.95$), smaller quota size (say, 30) actually incurs more overheads because of the more frequent quota allocation and de-allocation operations. These observations are supported by the data shown in Table 7.1.

To demonstrate the scalability of our PoQ dynamic bandwidth allocation scheme, we examine how the expected cost of link QoS state update per accepted flow changes as we increase the network capacity (in this case the bottleneck link capacity C). The results are plotted in Figure 7.9 for two different quota sizes (a) 30 and (b) 100. From the figures, we see that as the network capacity increases, the expected link level QoS update cost per accepted flow decreases. This is actually not too surprising (see our comments at the end of Section 7.3.2): with the normalized network load fixed, the probability that a flow is accepted in the critical mode decreases as the link capacity increases, due to the increased multiplexing gains. From these results we conclude that the PoQ scheme scales well as the network capacity increases. This is particularly the case, when the network is not heavily overloaded. When the network is heavily loaded, our scheme still leads to some amount of cost reduction (especially with appropriately chosen quota size), albeit not as significant as when the network is not heavily loaded. Note that when the network is heavily overloaded,



(a) Quota size = 30.



(b) Quota size = 100.

Figure 7.9: Expected cost of link QoS state updates as the network capacity increases.

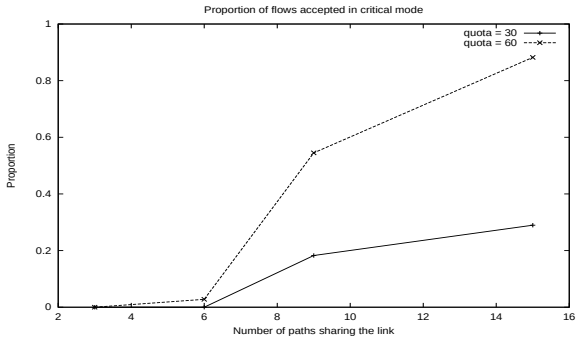


Figure 7.10: Proportion of flows accepted in critical mode as the number of paths increases (C = 5400, a = 0.95).

some slow-down in flow request processing may not be a severe problem, since the network itself is not capable of accommodating all the flows. Furthermore, in this case we can use an extended PoQ scheme (the lossy-path PoQ scheme introduced in the next section) to further improve the flow processing capability of the bandwidth broker.

Lastly, we consider the impact of the number of paths sharing a bottleneck link on the performance of the PoQ scheme. Figure 7.10 shows the proportion of flows accepted in critical mode as we increase the number of paths sharing the bottleneck link. In this set of simulations the normalized load a is set to 0.95. Note that when there are a small number of paths, most of the flows can be accepted in the normal mode. But when the number of paths are large, large quota size causes more flows to be accepted in the critical mode. This is because there are not enough quotas to go around among all the paths. As a general rule-of-thumb, in order to make the PoQ scheme work efficiently, the ratio of the number of quotas a link has over the number of the paths sharing the link should be reasonably large. In particular, a network with many paths sharing a bottleneck link, smaller quota sizes are preferred.

7.4 Multiple Bandwidth Broker Design

In this section we extend the centralized bandwidth broker architecture with a single bandwidth broker to a hierarchically distributed architecture with multiple bandwidth brokers. This multiple bandwidth broker (MBB) architecture addresses the scaling problem posed by the potential communication bottleneck between the bandwidth broker system and the edge routers. The MBB architecture is presented in Section 7.4.1, where an extended PoQ mechanism — the *lossy-path PoQ* dynamic bandwidth allocation scheme — is also in-

troduced to further reduce the call processing overheads at the central bandwidth broker. Simulation results are presented in Section 7.4.2.

7.4.1 The MBB Architecture and the Lossy-Path PoQ scheme

The *hierarchically distributed* multiple bandwidth broker architecture we propose is designed based on the two-level network QoS representation and the PoQ dynamic bandwidth broker architecture. As illustrated in Figure 7.11, the proposed MBB architecture consists of a *central* bandwidth broker (cBB) and a number of *edge* bandwidth brokers (eBBs). The central bandwidth broker maintains the link QoS state database and manages quota allocation and de-allocation among the edge bandwidth brokers. Whereas, each of the edge bandwidth brokers manages a *mutually exclusive* subset of the path QoS states and performs admission control for the corresponding paths. The number of eBBs can vary, depending on the size of the network domain. In the extreme case, for example, we can have one eBB for each edge router (as shown in Figure 7.11), and the eBB can co-locate at the edge router.

When a flow arrives at an edge router, the flow reservation set-up request is forwarded by the edge router to the eBB that is in charge of the flow's path. The eBB will make admission control based on the path state it maintains such as the currently available bandwidth allocated to the path. If no sufficient bandwidth is available on the path, the eBB requests a new quota for the path from the cBB. If the request is granted, the eBB admits the flow and updates its path QoS state. If the request fails (i.e., one or more links along the path are critically loaded), we can operate just like the basic PoQ scheme: the eBB forwards the flow reservation request to the cBB, which will perform admission control using the per-flow link-update scheme. We refer to this eBB operation model the *non-lossy-path* model, as no flows will ever be rejected by the eBB, based on its path QoS state. We now introduce an alternative eBB operation model — the *lossy path* model. Under this model, when a quota request fails, the eBB will simply reject the flow reservation request, instead of passing it to the cBB. We refer to the PoQ dynamic bandwidth allocation scheme under the lossy path model the *lossy-path PoQ* scheme. With the lossy-path PoQ scheme, the role of cBB is much simpler: it performs only quota management, and all admission control decisions are now delegated to the eBBs. Combining the proposed MBB architecture with this lossy-path PoQ scheme, we can not only avoid the communication bottleneck to the cBB, but also significantly reduce the processing burden at the cBB. This is particularly desirable in a large network with high traffic intensity. Clearly, the enhanced scalability of

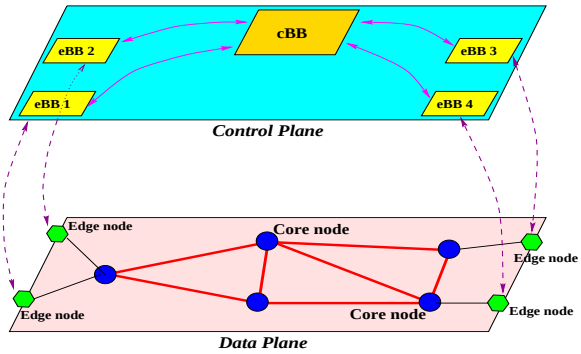


Figure 7.11: Multiple bandwidth brokers on the control plane for a network domain.

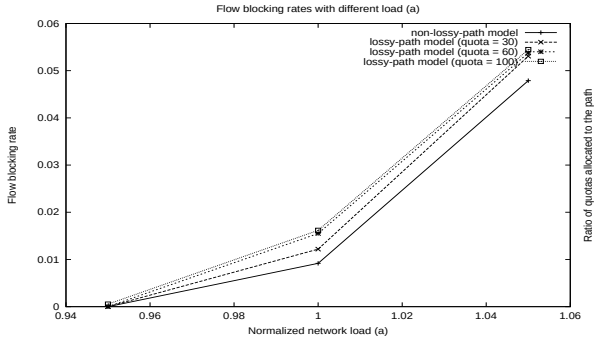


Figure 7.12: Flow blocking rates of the non-lossy-path and lossy-path models (C = 5400).

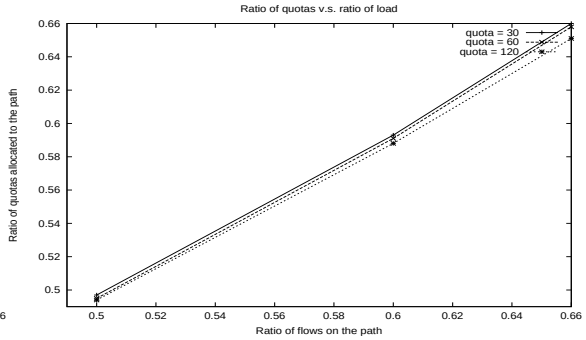


Figure 7.13: Quotas allocated to a path is proportional to the traffic load distributed on the path (C = 5400, a = 0.95).

the architecture is gained at the expense of some loss in performance, as some flows that are rejected may be accommodated if the non-lossy-path model is used. In the next section we will investigate the performance implication of the lossy-path MBB architecture model.

Before we move on to the simulation investigation, we would like to comment on some of the advantages of the proposed MBB architecture. Note that a straightforward approach to building a distributed bandwidth broker architecture to avoid the communication bottleneck problem would be a replicated bandwidth broker system, with multiple identical bandwidth brokers geographically dispersed in the network system. However, due to the need to both access and update the network QoS states, maintaining *consistent QoS state databases* requires synchronization among the bandwidth brokers, which can be time-consuming and problematic. In contrast, our hierarchically distributed bandwidth broker architecture does not suffer such a problem, owing to the appropriate partition of the path QoS states and the PoQ dynamic bandwidth allocation mechanism we employ.

7.4.2 Simulation Investigation

In this section, we conduct simulations to study the performance of the MBB architecture using the lossy-path PoQ dynamic bandwidth allocation scheme. We use the same network topology as shown in Figure 7.6, with the number of paths traversing the bottleneck link set to 3. We assume that there is an eBB associated with each path. The normalized link capacity with respect to the flow rate is 5400. All flows have an exponential holding time with a mean of 900 seconds. We again vary the flow arrival rate to produce different network loads.

Recall that under the lossy-path MBB architecture, an eBB will reject a flow when its request to the cBB for a new quota fails, i.e., when the bottleneck link has no quota left. Note that in this case, the total unreserved bandwidth on the link *may* be sufficient to accommodate the flow, since it is possible that not all the paths have used up the bandwidth allocated to it. Hence in general, the lossy-path model may result in a higher flow blocking rate than the non-lossy-path model. Figure 7.12 shows the flow blocking rates of the lossy-path model with three different quota sizes, as we vary the network load. The flow blocking rate of the non-lossy-path model is also plotted for comparison. We see that when the normalized network load is below 0.95, all flows are accepted under all the schemes. As the load is increased, a small portion of flows is rejected. The lossy-path model suffers some performance loss compared to the non-lossy-path model. The larger the quota size is, the bigger the performance loss is. In addition, the performance loss enlarges as the network load increases. However, after the normalized network load reaches 1, the performance loss does not seem to increase visibly, in particular for the two larger quota sizes. This is likely due to the fact that once the network is overloaded, a large portion of those flow reservation set-up requests that are forwarded by the eBBs to the cBB for admission control under the non-lossy-path model end up being rejected by the cBB. Hence rejecting these flow requests at the eBBs does not degrade the system performance significantly, in particular when the network is highly overloaded. Overall, we observe that the performance loss caused by the lossy-path model is fairly small. We believe that at the expense of a relatively small performance loss, the reduced bandwidth broker system overhead may well be worthwhile, in particular when the network system itself is overloaded.

Lastly, we show the importance of dynamic quota allocation used in our PoQ scheme in adapting to the offered load along a path. Dynamically allocating quotas to match the flow activity along a path is particularly important under the lossy-path model, as inadequate quota allocation can unnecessarily cause flows being rejected, and thus degrade the system

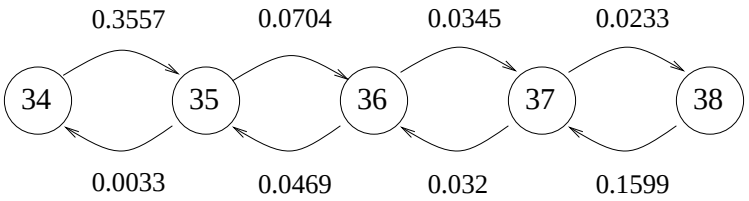


Figure 7.14: Quota state transition rate (C = 3600, a = 0.9, quota = 30).

performance. For this reason, we conduct simulations to evaluate the ability of the dynamic PoQ scheme to track the traffic load on a path. Figure 7.13 shows the ratio of quotas allocated to a path to the total quotas, as we increase the proportion of flows distributed onto the path. The normalized network load is fixed at 0.95. The results in this figure show that the dynamic quota allocation used in our scheme is indeed able to track the traffic intensity of a path very well.

7.5 Improvements on the Performance of the PoQ Scheme

As we discussed in Sections 7.3 and 7.4, the performance of the PoQ scheme depends critically on the quota management used in the scheme. For example, to control the impact of the scheme on the flow blocking rate of the system, smaller quotas are preferred; on the other hand, smaller quotas introduce more overhead into the system, because of potentially more frequent quota allocations and de-allocations. In this section, we propose several extensions to the quota management of the PoQ scheme, aiming to improve the performance and enhance the flexibility of the PoQ scheme under different environments. For each extension, we will first briefly describe the mechanism, and then conduct simulations to investigate the performance of the mechanism. These extensions can be applied to both the lossy-path and non-lossy-path models. In the simulation study presented here, however, we will mainly focus on the lossy-path model. Throughout this section, we use the same network topology as depicted in Figure 7.6 for the simulations, where the number of paths traversing the bottleneck link is set to three.

7.5.1 PoQ with Hysteresis

In the PoQ scheme we presented in Sections 7.3 and 7.4, a path will immediately return an excess quota to the links along the path whenever there is one available. In this way, the returned excess quota can be used by other paths which share a link with the path. However, it is possible that a path has an excess quota only because of *short-term, local*

flow fluctuations. Therefore, the path may need to request a quota shortly after it returns an excess quota to the links.

To have a better understanding of this phenomenon, we introduce the notion of *quota state transition rate* of a path as follows. We say a path is in a quota state i if currently i quotas are allocated on the path. Let T_i denote the total time that a path stays in a quota state i in an arbitrary time interval, let $N_{i,j}$ be the number of transitions from quota state i to state j during the same time interval. Then the quota state transition rate $qstr_{i,j}$ of the path from state i to j in the time interval is defined as,

$$qstr_{i,j} = \frac{N_{i,j}}{T_i}. \tag{7.3}$$

Figure 7.14 shows the quota state transition rates of the path $I1 \rightarrow E1$ after warm-up of the simulation, where the capacity of the bottleneck link is set to 3600, the normalized network load is 0.9, and the quota size is 30. All flows have an exponetial holding time with a mean of 900 seconds. From the figure we see that the quota state transition rates from the state 34 to 35 and from the state 38 to 37 are much higher than the transition rates between other states. A close examination of the trace data reveals that most of the time the path has 35, 36 or 37 quotas and only occasionally goes to the states 34 and 38. Moreover, when the path quota state goes to 34 or 38, it will only stay there in a *very short* time.

Such a short time state visit is caused by the short-term local flow fluctuations where a small number of consecutive flow arrivals and departures triggers both a quota allocation and a de-allocation in a very short time. Note that short-term local flow fluctuations also occur in the state transitions between 35, 36 and 37 even though they are not exposed in this simple figure.

Recall that the operations of quota allocations and de-allocations involve *per link* QoS state updates along the path, which increases the overall overhead of the system. Therefore, we should limit the amount of quota allocations and de-allocations as small as possible. One simple strategy maybe just allow a path to hold excess quotas without returning them. However, this may cause higher flow blocking rates on the other paths which share a link with the path because of the shortage of available quotas, which is undesirable.

To regulate the behavior of a path in handling excess quotas, we develop the following simple mechanism based on the notion of *hysteresis*: Each path will maintain a threshold to determine if the path should return an excess quota. Instead of returning an excess quota

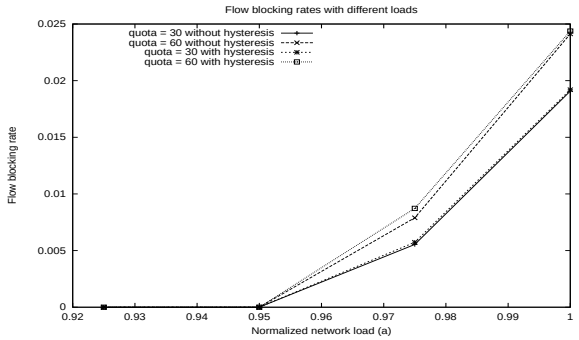


Figure 7.15: Effects of hysteresis on flow blocking rates (C = 3600, hysteresis = 0.05).

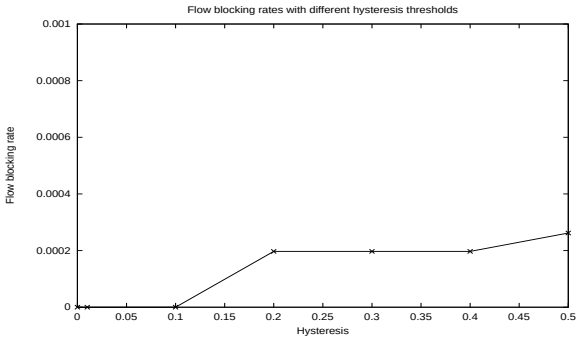


Figure 7.16: Flow blocking rates with different hysteresis thresholds (C = 3600, a = 0.95, quota = 60).

immediately after it is available, a path will only return an excess quota after the reservation rate along the path is below some threshold with respect to the current allocated quotas on the path. More formally, let $C^{\mathcal{P}}$ denote the bandwidth allocated on a path $\mathcal{P}$, let $R^{\mathcal{P}}$ be the total reservation rate of flows along the path. Then the path will only return an excess quota if,

$$R^{\mathcal{P}} \leq C^{\mathcal{P}} - (1 + hysteresis)B^{quota}, \tag{7.4}$$

where B^{quota} is the bandwidth of a quota, and $hysteresis \geq 0$.

The intuition of the hysteresis based scheme is quite straightforward. Instead of returning an excess quota whenever it is available, the hysteresis based scheme ties the policy of quota de-allocation of a path to the (flow) traffic load on the path. When the reservation rate on a path is below some threshold, it is possible that the path will not use the excess quota in a relatively long time interval. Therefore, it is desirable to release the excess quota so that it can be re-used by other paths.

In the following, we conduct simulations to study the effectiveness of the hysteresis based scheme. The normalized capacity of the bottleneck link with respect to the flow rate is 3600. All flows have an exponential holding time with a mean of 900 seconds. We vary the flow arrival rate to produce different network loads. In these simulations, the value of $hysteresis$ is set to 0.05.

Figure 7.15 shows the flow blocking rates of the hysteresis based PoQ scheme under the lossy-path model for two quota sizes: 30 and 60. For comparison, we also include the

Table 7.2: Effects of hysteresis on quota allocations and deallocations (C = 3600, hysteresis = 0.05)

	PoQ without hysteresis				PoQ with hysteresis			
quota size	30		60		30		60	
normalized load	0.925	1.000	0.925	1.000	0.925	1.000	0.925	1.000
quotas allocated	517	424	361	123	168	167	92	43
quotas deallocated	523	424	364	123	173	167	94	43

corresponding curves of the PoQ scheme without hysteresis. From the figure we see that when the normalized network load is below 0.95, no flow gets rejected no matter hysteresis based mechanism is used or not. In these cases, the quota-based bandwidth allocation scheme actually does not affect the flow blocking rate because of the low network load. As the network load increases, the results are different for different quota sizes. When the quota size is 30, the effects of the hysteresis based scheme on the flow blocking rate is minor. There are two possible reasons. First, the quota size (30) is relatively small, the major contributor to the flow blocking rate is the network load instead of the quota allocation scheme. Second, when the quota size is 30, the threshold ($(1 + hysteresis)B^{quota}$) for the PoQ scheme *with* hysteresis to return an excess quota is 31.5, which is close to the quota size (30), the *threshold* for the PoQ scheme *without* hysteresis to return an excess quota. Therefore the extra bandwidth that may be *wasted* by the hysteresis scheme is small (1.5). When the quota size is 60, however, the hysteresis based scheme causes a slightly higher flow blocking rate because of the relatively larger hysteresis threshold.

As discussed above, the motivation of the hysteresis based mechanism is to reduce the amount of quota allocations and de-allocations of the PoQ scheme. Table 7.5.1 presents the corresponding quota allocation and de-allocation activities after the warm-up of the simulations. From the table we see that the reductions are significant by such a hysteresis based mechanism. Compared with PoQ without hysteresis, there are roughly 3 to 4 times reduction in the number of quota allocations and de-allocations in the hysteresis based scheme.

Clearly, the performance and effectiveness of the hysteresis based scheme relies on the value of $hysteresis$ of a path. If the value is too conservative, more flows may get rejected on other paths which share a link with the path because of the shortage of available quotas. On the other hand, a too optimistic value may undermine the scheme because the threshold is not large enough to hold an excess quota which the path will need shortly after it is available. Therefore, high frequent quota allocations and de-allocation may still occur.

Figure 7.16 shows the flow blocking rate of the system for a case where the quota size is 60, and Table 7.5.1 presents the corresponding quota allocation and de-allocation activities of the simulations. Note that when the value of $hysteresis$ is 0.01, the hysteresis based scheme has the same flow blocking rate (i.e., 0) as the PoQ scheme without hysteresis ($hysteresis = 0$). However, as we can see from the table, when $hysteresis = 0.01$ the amount of quota allocations and de-allocations is still quite high. As the value of $hysteresis$ increases to 0.2, the number of quota allocations and de-allocations drops dramantically with a slight increase in the flow blocking rate of the system. However, even we further increase the value of $hysteresis$, the reduction of the amount of quota allocations and de-alloctions is little. From the above discussion we can see that a modest value of $hysteresis$ should work well in balancing the reduction of the amount of quota allocations and de-allotions and the increase in the flow blocking rate of the system.

Note that when the values of $hysteresis$ are 0.2, 0.3 and 0.4, the curve of the flow blocking rate of the system is flat. This is partly caused by the fact that the threhold with $hysteresis = 0.2$ has caught the trend in the flow fluctuation in a degree, that is, when the reservation rate of a path is below the threshold with $hysteresis = 0.2$, it will soon be below the threshold with $hysteresis$ equal to 0.3 or 0.4. Therefore, they all have the similar effects on the quota de-alloctions (see also Table 7.5.1) , and hence the similar flow blocking rates.

So far, we have discussed the employment of the hysteresis based approach for a path to hold an excess quota to accommodate the short-term flow fluctuations. Note that it may be desirable for a path to employ a *forward threshold* to request an extra quota before the path has used up all the allocated quotas on the path [38], especially in the multiple BB architecture. By such a forward threshold, the path may handle flow requests immediately when the requests come instead of waiting for a new quota if the allocated quota bandwidth has been used up on the path. We leave it as an engineering choice and will not elaborate on this in this chapter.

Table 7.3: Effects of different hystereses on quota allocations and deallocations (C = 3600, a = 0.95, quota = 60)

hysteresis	0	0.01	0.1	0.2	0.3	0.4	0.5
quotas allocated	172	87	30	16	15	15	13
quotas deallocated	173	88	31	18	17	17	15

7.5.2 PoQ with Variable Quota Size

Allocating bandwidth based on the notion of quota instead of per flow bandwidth requests provides us with an efficient bandwidth allocation and admission control mechanism. However, As shown in Sections 7.3 and 7.4, the performance and cost of the scheme is closely related to the size of the quota used in the scheme. Generally speaking, smaller quotas are preferred as the flow blocking rate concerns; on the other hand, to reduce the amount of quota allocations and de-allocations, larger quotas are favored. In this section, we propose a scheme based on the notion of *variable size* quota to overcome the dilemma on how to choose a proper quota size.

Let $\delta_1, \delta_2, \ldots, \delta_N$ denote the quota sizes that will be used in a network domain. Without loss of generality, we assume that $\delta_1 > \delta_2 > \ldots > \delta_N$. Corresponding to each δ_i, there is a link bandwidth allocation threshold θ_i, $i = 1, 2, \ldots, N$, where $\theta_i < \theta_{i+1}$ for $i = 1, 2, \ldots, N-1$. Let C denote the capacity of a link, B the current total bandwidth allocated to all the paths traversing the link. We define the quota allocation mechanism as follows: A quota of size δ_i is allocated when there is a quota allocation request if $\theta_{i-1} < \frac{B}{C} \leq \theta_i$ for $i = 1, 2, \ldots, N$, where we used the convention that $\theta_0 = 0$.

Figure 7.17 shows the flow blocking rates of the variable quota size scheme (VQS), where the link capacity is set to 3600 with respect to the flow rate. In these simulations, the quota bandwidth allocation thresholds are 0.9 and 1. For example, when the configured quota sizes are 60 and 30 in a network domain, a link will allocate a quota of size 60 to meet a quota request if the proportion of the link bandwidth that has been allocated is less than or equal to 0.9. Otherwise, a quota with size 30 will be allocated. For comparison, the flow blocking rates of the original PoQ uniform quota size (UQS) bandwidth allocation scheme are also included with a quota size 60 and 30, respectively.

Note that the curve of the flow blocking rate of VQS with quota sizes 60 and 30 is almost identical with the curve of UQS with a quota size 30. These results are not surprising because, when the allocated bandwidth goes beyond 90 percent of the link capacity, the smaller quota size is used under VQS, which is equal to the quota size used in the corresponding UQS.

To examine the advantage of the variable quota size scheme in quota allocation, we present the quota allocation and de-allocation activities in Table 7.5.2 for both VQS and UQS. For VQS, the quota sizes are 60 and 30, while for UQS the quota size is 30.

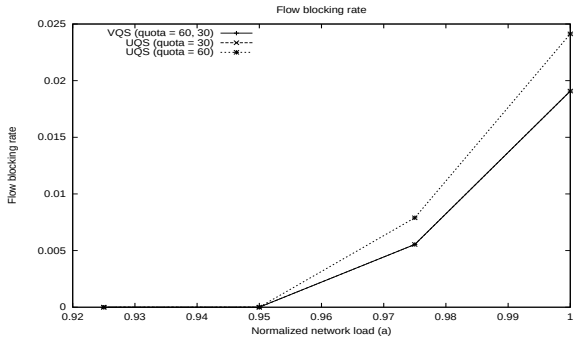


Figure 7.17: Effects of variable quota size on flow blocking rates (C = 3600, θ_1 = 0.9).

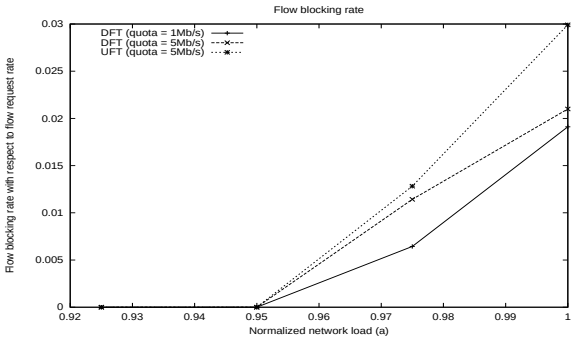


Figure 7.18: Flow blocking rate with different flow treatment (C=3000).

As shown from the table, when the normalized network load is 0.9, the VQS scheme has a relatively smaller amount of quota allocations and de-allocations compared with the UQS scheme. As the network load increases, both schemes have the almost identical amount of quota allocations and de-allocations. Note that the threshold for the VQS scheme to allocate a quota of size 30 instead of 60 is 0.9. Therefore, when the network load is below or equal to 0.9, a quota with larger size (60) is allocated each time under the VQS scheme. Hence the VQS scheme has a relatively small amount of quota allocations and de-allocations in these cases. As the network load increases, the VQS scheme will allocate a quota of smaller size (30), which is identical to the quota size in the UQS scheme. Therefore, they have almost the same amount of quota allocations and de-allocations when the network load is beyond the quota bandwidth allocation threshold (0.9).

Table 7.4: Effects of the variable quota size scheme on quota allocations and de-allocations (C = 3600, θ_1 = 0.9)

quota size	quota = 30 (UQS)					quota = 60, 30 (VQS)				
normalized load	0.900	0.925	0.950	0.975	1.000	0.900	0.925	0.950	0.975	1.000
quotas allocated	483	517	538	523	424	419	510	536	523	424
quotas deallocated	483	523	540	524	424	419	516	538	524	424

From the above discussion and the simulation results, we note that by using variable quota sizes, the VQS scheme has more freedom to allocate large quota sizes when the network load is low. When the network load becomes higher, it can start to behave more conservative on bandwidth allocation. In this way, it may reduce the cost of the system meanwhile maintaining a smaller flow blocking rate.

7.5.3 PoQ with Differentiated Flow Treatments

In both lossy-path and non-lossy-path models we have studied, flows are treated identically based the flow requests and the availability of the resources. For example, in the lossy-path model, as soon as there is no quota available at the central bandwidth broker, a flow will be rejected. However, under certain environments it may be desirable to treat flows differently based on some pre-defined policies. For example, some types of flows may be critical to the revenue of an ISP, therefore, instead of rejecting these flows at an eBB because of the shortage of the quotas, the eBB may forward the flow requests to the cBB, and the cBB can conduct the admission control for these flows based on different rules.

To illustrate the idea of the differentiated flow treatments with the PoQ scheme, we define two types of flows. One is the long-term *large* flows, another is the short-term *small* flows. A large flow normally requires a large amount of bandwidth, and a small flow requests a relatively small amount of bandwidth. As an example, we may consider video applications as large flows, while audio applications like IP Telephony as small flows.

Instead of treating large flows at the edge BBs, an eBB will forward these flow requests to the central BB. The cBB will conduct the admissibility test using the *link-update* admission control scheme, that is, no large flows will be rejected as soon as there is enough bandwidth along the path. We call such a scheme as the PoQ with differentiated flow treatment (DFT), and the original lossy-path scheme as the PoQ with uniform flow treatment (UFT). Note that, the PoQ with DFT scheme can be considered as a hybrid of the lossy-path and non-lossy-path models: treating small flows with the lossy-path model while treating large flows with the non-lossy-path model. Moreover, because large flows are forwarded to the cBB, a smaller quota size can be used by the cBB to meet the quota requests from the eBBs. Recall that as the flow blocking rate concerns, a smaller quota size is preferred, therefore, we may have lower flow blocking rate by such a DFT scheme.

We conduct a set of simulations to study the property of the PoQ scheme with DFT. In these simulations, small flows have a request rate uniformly chosen in a range from 16 Kb/s to 64 Kb/s; large flows have a request rate uniformly chosen in another range from 0.5 Mb/s to 1.5 Mb/s. The mean holding time for a small flow is 900 seconds, while for a large flow, it is 1800 seconds. We vary the interarrival times of flows to produce different network load, however, the ratio of the offered load of the small flows over that of the large flows is set to $9/1$. Therefore the overall per flow mean rate is 136 Kb/s. The normalized link capacity is 3000 with respect to the mean flow rate (136 Kb/s). All the simulations last 15000 seconds,

of which the first 10000 seconds are the warm-up times.

To accommodate the effect of different rate requirements of small and large flows, we define the flow blocking rate *with respect to flow request rates* as follows.

$$\Upsilon = \frac{\sum_{i \in \mathcal{F}_1} r_i}{\sum_{i \in \mathcal{F}_2} r_i}, \tag{7.5}$$

where $\mathcal{F}_1$ is the set of flows rejected, $\mathcal{F}_2$ the set of all the flows during the simulation, and r_i the requested rate of a flow i.

Figure 7.18 shows the flow blocking rates with respect to flow request rates for the PoQ with DFT and PoQ with UFT schemes. From the figure we see that, when the quota size is 5 Mb/s, UFT has a higer flow blocking rate compared to the DFT scheme. Recall that DFT can be considered as a hybrid scheme of the lossy-path and non-lossy-path models, while the UFT is a lossy-path model. Therefore, a flow is rejected in the UFT scheme whenever there is no quota available, but a (large) flow may still get accepted even there is no quota available by the *link-update* admission control scheme in the DFT model. As we discussed earlier, a quota of size 1.5M b/s should be the smallest quota size that an UFT scheme can use because large flows may have a request rate equal to 1.5M b/s. On the other hand, the DFT scheme may choose a relatively small quota size (quota = 1M b/s) to have a smaller flow blocking rate, as illustrated in the figure.

7.6 Summary

In this chapter we studied the scalability issue in the design of a centralized bandwidth broker model for dynamic control and management of QoS provisioning. We identified two major factors that may potentially affect the scalability of the centralized bandwidth broker architecture: the memory and disk access speed and communication capacity between the bandwidth broker and edge routers. To reduce the overall number of QoS state accesses and updates, we developed a path-oriented quota-based (PoQ) dynamic bandwidth allocation mechanism for efficient admission control operations under the centralized bandwidth broker model. Based on the proposed dynamic bandwidth allocation mechanism, we also extended the centralized bandwidth broker architecture to a hierarchically distributed architecture with multiple bandwidth brokers to address the scaling problem posed by the potential communication bottleneck between the bandwidth broker system and edge routers.

Our simulation investigation demonstrated that the proposed PoQ dynamic bandwidth allocation mechanism is indeed an effective means to increase the overall call processing capability of the bandwidth broker. Furthermore, the bandwidth broker architecture can be designed in such a manner that it scales with the increase in the network capacity. Further extensions to the PoQ scheme were also investigated to improve the performance of the PoQ scheme and to enhance the flexibility of the bandwidth broker architecture.

Part III

Service Overlay Networks

Chapter 8

Bandwidth Provisioning for Service Overlay Networks

8.1 Introduction

Today's Internet infrastructure supports primarily *best-effort connectivity* service. Due to historical reasons, the Internet consists of a collection of network domains (i.e., autonomous systems owned by various administrative entities). Traffic from one user to another user typically traverses multiple domains; network domains enter various bilateral business relationships (e.g., provider-customer, or peering) for traffic exchange to achieve global connectivity. Due to the nature of their business relationships, a network domain is only concerned with the network performance of its own domain and responsible for providing service guarantees for its customers. As it is difficult to establish multi-lateral business relationship involving multiple domains, deployment of end-to-end services beyond the best-effort connectivity that requires support from multiple network domains is still far from reality. Such problems have hindered the transformation of the current Internet into a truly multi-service network infrastructure with end-to-end QoS support.

We propose and advocate the notion of *service overlay network* (SON) as an effective means to address some of the issues, in particular, end-to-end QoS, plaguing the current Internet, and to facilitate the creation and deployment of *value-added Internet services* such as VoIP, Video-on-Demand, and other emerging QoS-sensitive services. The SON network architecture relies on well-defined business relationships between the SONs, the underlying network domains and users of the SONs to provide support for end-to-end QoS: the SON

purchases bandwidth with certain QoS guarantees from the individual network domains via *bilateral service level agreement (SLA)* to build a logical end-to-end service delivery infrastructure on top of the existing data transport networks; via a service contract (e.g., a usage-based or fixed price service plan), users[1] directly pay the SON for using the value-added services provided by the SON.

Figure 8.1 illustrates the SON architecture. The SON is pieced together via *service gateways* which perform service-specific data forwarding and control functions. The *logical* connection between two service gateways is provided by the underlying network domain with certain bandwidth and other QoS guarantees. These guarantees are specified in a bilateral SLA between the SON and the network domain. This architecture, for example, bypasses the peering points among the network domains, and thus avoids the potential performance problems associated with them. Relying on the bilateral SLAs the SON can deliver end-to-end QoS sensitive services to its users via appropriate provisioning and service-specific resource management.

In addition to its ability to deliver end-to-end QoS sensitive services, the SON architecture also has a number of other important advantages. For example, it decouples application services from network services, thereby reducing the complexity of network service management and control, especially in terms of QoS management and control. The network domains are now concerned primarily with provisioning of data transport services with associated bandwidth management, traffic engineering and QoS guarantees on a much coarser granularity (per SON). In particular,the notion of SON also introduces a new level of traffic aggregation – *service aggregate*: the underlying network domains can aggregate traffic based on the SONs they belong to and perform traffic and QoS control accordingly based on the corresponding SLAs. Under the SON architecture, a SON is responsible for ensuring end-to-end QoS for its services. Because of its service awareness, a SON can deploy service-specific provisioning, resource management and QoS control mechanisms (e.g., at service gateways) to optimize its operations for its services. Hence the SON architecture not only simplifies the network QoS management and makes it more scalable, but also enables flexible creation and deployment of new (value-added) services.

Obviously deployment of SON is a capital-intensive investment. It is therefore imperative to consider the *cost recovery* issue for the SON. Among the many costs the SON deployment incurs (e.g., equipment such as service gateways), a dominant *recurring* cost is the cost of bandwidth that the SON must purchase from the underlying network domains to

[1]Users may also need to pay (i.e., a monthly fee) the access networks for their right to access the Internet.

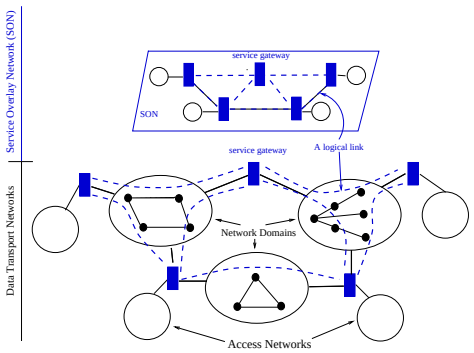


Figure 8.1: An illustration of a service overlay network.

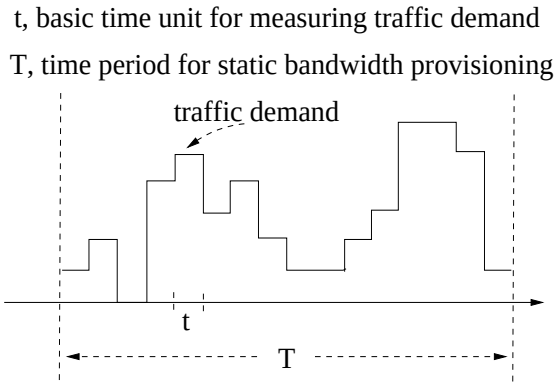


Figure 8.2: Traffic demands.

support its services. The SON must provision adequate bandwidth to support its end-to-end QoS-sensitive services and meet traffic demands while minimizing the bandwidth cost so that it can generate sufficient revenue to recover its service deployment cost and stay profitable. *The bandwidth provisioning problem is therefore a critical issue in the deployment of the SON architecture.* This chapter is devoted to the study of this issue.

We develop analytical models to study the problem of SON bandwidth provisioning and investigate the impact of various factors on SON bandwidth provisioning: SLAs, service QoS, bandwidth costs and traffic demands. We consider the so-called *pipe* SLA model as an example to illustrate how the SON bandwidth provisioning problem can be formally defined. The analyses and solutions can be adapted to the so-called *hose* SLA model. In Section 8.2 we describe how the SON logical topology can be represented under the pipe SLA model and present the model assumptions. Using the pipe SLA model we present a basic static SON bandwidth provisioning solution in Section 8.3, and study the problems of the more general *static* and *dynamic* SON bandwidth provisioning, respectively in Section 8.4 and Section 8.5. Analytical models and approximate solutions are developed for both static and dynamic bandwidth provisioning. Numerical studies are also performed to illustrate the properties of the proposed solutions and demonstrate the effect of traffic demand distributions and bandwidth costs on SON bandwidth provisioning.

The notion of overlay networks has been used widely in telecommunication and data networks. For example, more recently content distribution networks and application layer multicast networks have been used for multimedia streaming [12]; *Detour* [67] and *Resilient Overlay Network* (RON) [1] employ the overlay technique to provide better routing support. Moreover, the overlay technique has attracted a lot of attention from industry [16, 17] as a means to deliver diverse QoS-sensitive services over the Internet. The service over-

lay networks we propose here is simply a generalization of these ideas. Perhaps what is particularly interesting is the use of SONs to address end-to-end QoS deployment issue. The major contribution of our work however lies in the study of the SON bandwidth provisioning problem. Our approach and formulation also differ from the traditional capacity planning in telephone networks (e.g. [35, 48]) in that we explicitly take into account various factors such as SLAs, QoS, traffic demand distributions.

8.2 Assumptions and Bandwidth Provisioning Problems

In this section we first describe a logical topology representation of the SON under the pipe SLA model and a simplifying assumption on service QoS. Two modes of bandwidth provisioning—*static* and *dynamic* bandwidth provisioning—that we will study in this chapter is then introduced. We conclude this section by describing the traffic demand model and a few notations regarding service revenue and bandwidth cost that will be used later in this chapter.

8.2.1 SON and Service QoS

The pipe SLA model is a common SLA model used in today's Internet. Under the pipe model, the SON can request bandwidth guarantees between any two service gateways across a network domain (see Figure 8.1); in other words a "pipe" with certain bandwidth guarantee is provisioned between the two service gateways across the network domain. To emphasize the relationship between the service gateways and the underlying network domains, we denote the *logical* (uni-directional) connection from a service gateway u to a neighboring service gateway v across a network domain D by $\langle u, v; D\rangle$, and refer to it as a *logical link* (or simply a *link*) between u and v across D. Note that between the SON and access networks where traffic to the SON originate and terminate, the *hose* SLA model is assumed to be used where certain amount of bandwidth is reserved for traffic *entering* or *exiting* the SON. We can treat each access network A as a *fictitious* service gateway u_A. Then we can talk about "connection"between u_A and a neighboring service gateway v across A and the corresponding "logical link" $\langle u_A, v; A\rangle$.

Given a logical link $l = \langle u, v; D\rangle$, the SON provider will contract with the network domain D to provide a certain amount of bandwidth guarantee c_l between the service gateways u and v across D. The SON bandwidth provisioning problem is then to determine how much bandwidth to be provisioned for each link $l = \langle u, v; D\rangle$ so that: 1) the end-to-end

QoS required by its services can be supported adequately; and 2) its overall revenue or net income can be maximized.

Although the QoS a SON must support for its services can be quite diverse (e.g., bandwidth, delay or delay jitter guarantees), in almost all cases a key component in providing such guarantees is to exert some form of control on the link utilization level, i.e., to ensure the overall load on a link does not exceed some specified condition. In other words, for the purpose of bandwidth provisioning, we assume that it is possible to map the service QoS guarantee requirements to a link utilization threshold[2]. To state this assumption formally, we assume that a link utilization threshold η_l is specified for each link l; and to ensure service QoS, the bandwidth c_l provisioned for link l must be such that the (average) link utilization stays below η_l.

8.2.2 Bandwidth Provisioning Modes

We consider two modes of bandwidth provisioning under the pipe model: *static* bandwidth provisioning and *dynamic* bandwidth provisioning. In *static* bandwidth provisioning mode, a SON contracts and purchases a fixed amount of bandwidth *a priori* for each pipe connecting the service gateways from the underlying network domains. In other words, the bandwidth is provisioned for (relatively) long period of time without changing. In *dynamic* bandwidth provisioning mode, in addition to the ability to contract and purchase bandwidth for each pipe *a priori*, a SON can also *dynamically* request for additional bandwidth from the underlying network domains to meet its traffic demands, and pay for the dynamically allocated bandwidth accordingly. To account for the potential higher cost in supporting dynamic bandwidth provisioning, it is likely that the underlying network domains will charge the SON different prices for statically provisioned and dynamically allocated bandwidth. Hence in either mode the key question in bandwidth provisioning is to determine the appropriate amount of bandwidth to be purchased *a priori* so that the total overall net income of a SON is maximized while in the meantime meeting the traffic demands as well as maintaining the service QoS.

[2]This particularly will be the case if the underlying network domain employs aggregate packet scheduling mechanisms such as FIFO or priority queues. For example, it has been shown [11, 45, 87] that in order to provide end-to-end delay guarantees, link utilization must be controlled at a certain level. Hence from the bandwidth provisioning perspective we believe that this assumption on service QoS is not unreasonable in practice. In fact it is said that many of today's network service providers use a similar utilization based rule (e.g., an average utilization threshold of 60% or 70%) to provision their Internet backbones.

8.2.3 Traffic Demand, Service Revenue and Bandwidth Cost

We now describe the traffic demand model for the SON. Recall that we assume that traffic always originates from and terminates at access networks. Given a source node s and destination node d, for simplicity we assume that a fixed route r consisting of a series of links connecting s and d is used to forward traffic from s to d. Let R denote the collection of routes between the source and destination nodes. Then the traffic demands over the SON can be represented by the traffic demands over these routes: for each $r \in R$, let ρ_r denote the (average) traffic demand (also referred to as traffic load) along route r measured over some period of time t (see Figure 8.2). The period t is relatively short, for example in seconds or several minutes, compared to the time scale of static bandwidth provisioning, denoted by T, which could be in several hours or days. The period t is considered as the basic unit of time. The set $\{\rho_r : r \in R\}$ then represents the traffic demands over the SON during the time unit they are measured, and is referred to as the traffic demand matrix of the SON. Note also that the traffic demands are always measured in units of bandwidth.

To capture the traffic demand fluctuations over time, we assume that the traffic demand ρ_r along each route r varies according to some distribution[3]. We denote the probability density function of the traffic demand distribution of ρ_r by $d\rho_r$. Then the probability that the traffic demand ρ_r exceeds x units of bandwidth is given by $\int_x^\infty d\rho_r$. Let $\bar{\rho}_r = \int_0^\infty \rho_r d\rho_r$, i.e., $\bar{\rho}_r$ is the (long-term) average traffic demand along route r over the time period for static bandwidth provisioning. Furthermore, we assume that the traffic demand distributions along the different routes are *independent*. In this chapter, we will study the bandwidth provisioning problem by considering two different traffic demand models. The first one takes into account the widely observed self-similar property of the Internet traffic by employing the $M/G/\infty$ input model [61, 63]; the second is based on the measurements of real Internet traffic.

For each route r, we assume that the SON receives e_r amount of revenue for carrying one unit of traffic demand per unit of time along route r. On the other hand, for each logical link or pipe l connecting two service gateways, the SON must pay a cost of $\Phi_l(c_l)$ per unit of time for reserving c_l amount of bandwidth from the underlying network domain. We refer to Φ_l as the bandwidth cost function of link l. Without loss of generality, we assume that Φ_l is a *non-decreasing* function.

[3]This traffic demand distribution can be obtained, for example, through long-term observation and measurement.

8.3 Basic Static Bandwidth Provisioning Model

In static bandwidth provisioning, a certain amount of bandwidth *overprovisioning* is needed to accommodate some degree of fluctuation in traffic demands. The key challenge in static bandwidth provisioning is therefore to decide the *optimal* amount of bandwidth overprovisioning. In this section, we present a basic static bandwidth provisioning model and analyze its properties. This basic model will serve as the basis for other bandwidth provisioning models we will consider in this chapter.

In the basic model, the SON provisions bandwidth on each link based on the long-term average traffic demand matrix $\{\bar{\rho}_r\}$, and attempts to maximize the *expected* net income. To accommodate some degree of fluctuation from the long-term average traffic demands, we introduce an *overprovisioning parameter* ϵ_l on each link l, $\epsilon_l \geq 0$. The meaning of the overprovisioning parameter ϵ_l is given as follows: we will provision c_l amount of bandwidth on link l such that as long as the overall traffic load on link l does not exceed its long-term average load by ϵ_l, the service QoS can be maintained, i.e., the link utilization is kept below the prespecified threshold η_l. To put it formally, define $\bar{\rho}_l = \sum_{r:l\in r} \bar{\rho}_r$, where $l \in r$ denotes that link l lies on route r. Then

$$\bar{\rho}_l(1+\epsilon_l) = (1+\epsilon_l) \sum_{r:l\in r} \bar{\rho}_r \leq \eta_l c_l, \forall l \in L \tag{8.1}$$

where L is the set of all links of the SON.

Given that c_l amount of bandwidth is provisioned on each link l, the expected net income of the SON is $\bar{W} = \sum_{r\in R} e_r \bar{\rho}_r - \sum_{l\in L} \Phi_l(c_l)$. Hence the basic bandwidth provisioning problem can be formulated as the following optimization problem:

$$\max_{c_l: l\in L} \bar{W} \quad \text{subject to (8.1).}$$

Since Φ_l's are non-decreasing, it is easy to see that the optimal solution to the optimization problem is given by

$$c_l^* = (1+\epsilon_l)\bar{\rho}_l/\eta_l \;\; \forall l \in L. \tag{8.2}$$

Hence under the basic bandwidth provisioning model, once we fix the overprovisioning parameters, the optimal amount of bandwidth to be provisioned for each link can be derived using (8.2).

Assuming that Φ_l's are sub-additive, we see that a sufficient condition for the SON to have positive expected net income is to ensure that

$$e_r > \frac{\sum_{l \in r} \Phi_l(c_l^*)}{\bar{\rho}_r} = \frac{\sum_{l \in r} \Phi_l(\frac{\bar{\rho}_r(1+\epsilon_l)}{\eta_l})}{\bar{\rho}_r}. \tag{8.3}$$

The relationship (8.3) provides a useful guideline for the SON to determine how it should set its price structure for charging users of its services to recover its cost of bandwidth provisioning. It has a simple interpretation: we can regard $\frac{\Phi_l(\bar{\rho}_r(1+\epsilon_l)/\eta_l)}{\bar{\rho}_r}$ as the average cost of carrying one unit of traffic demand per unit of time along route r on link l. Then the right-hand side of (8.3) is the total cost of carrying one unit of traffic demand per unit of time along route r. To recover its cost, the SON must then charge users of its services more than this amount. If Φ_l's are strictly concave (i.e., non-linear), in other words, the per-unit bandwidth cost decreases as the amount of reserved bandwidth increases, the economy of scale will benefit the SON: the higher the average long-term traffic demands, the lower the average cost of providing its services, yielding higher net income. In the case Φ_l's are linear, i.e., $\Phi_l(c_l) = \phi_l c_l$, then (8.3) becomes $e_r > \sum_{l \in r} \phi_l(1 + \epsilon_l)/\eta_l$ which is independent of the traffic demands.

8.4 Static Bandwidth Provisioning with Penalty

In the basic static bandwidth provisioning model we assume that the overprovisioning parameters are given. We now consider the problem of how to obtain the *optimal* overprovisioning parameters under given traffic demand distributions. We study this problem by taking into account the consequence of potential QoS violation when the actual traffic demands exceed the target link utilization. For this purpose, we assume that *the SON may suffer a penalty when the target utilization on a link is exceeded, and therefore service QoS may potentially be violated.* For example, it is quite likely that the service contract between the SON and its user is such that when the service QoS is poor (e.g., due to network congestion), a lower rate is charged, or the user may demand a refund. In the case that some form of admission control is used by the SON to guide against possible QoS violation, the penalty can be used to reflect the lost revenue due to declined user service requests. We will refer to this model as the *static bandwidth provisioning with penalty model*, or in short, *static-penalty* model.

For each route r, let π_r denote the average penalty suffered by per unit of traffic demand per unit of time along route r *when the service QoS along route* r *is potentially violated.*

Given the traffic demand matrix $\{\rho_r\}$, let $B_r(\{\rho_r\})$ denote the probability that the service QoS along route r is potentially violated, more specifically, *the target utilization on one of its links is exceeded*. Then the total net income of the SON for servicing the given traffic demand matrix $\{\rho_r\}$ can be expressed as follows:

$$W(\{\rho_r\}) = \sum_{r \in R} e_r \rho_r - \sum_{l \in L} \Phi_l(c_l) - \sum_{r \in R} \pi_r \rho_r B_r(\{\rho_r\}), \tag{8.4}$$

where in the above we use $W(\{\rho_r\})$ to emphasize the dependence of the total net income on the traffic demand matrix $\{\rho_r\}$. When there is no confusion, we will drop $\{\rho_r\}$ from the notation.

Let $d\{\rho_r\}$ denote the joint probability density function of the traffic demand matrix $\{\rho_r\}$, where recall that $d\rho_r$ is the probability density function of the traffic demand ρ_r along route r. Then the expected net income of the SON under the traffic demand distributions $\{d\rho_r\}$ is given by

$$E(W) = \int \cdot \int_{\{\rho_r\}} W(\{\rho_r\}) d\{\rho_r\}, \tag{8.5}$$

where $\int \cdot \int_{\{\rho_r\}}$ denotes multiple integration under the joint traffic demand distribution $\{d\rho_r\}$.

Now we can state the problem of static bandwidth provisioning with penalty as the following optimization problem: finding the optimal overprovisioning parameters $\{\epsilon_l\}$ to maximize the expected net income, i.e.,

$$\max_{\{\epsilon_l\}} E(W) \text{ subject to (8.1).} \tag{8.6}$$

Unfortunately, the exact solution to this optimization problem is in general difficult to obtain. It depends on both the particular forms of the traffic demand distributions $\{d\rho_r\}$ and the service QoS violation probabilities B_r. To circumvent this difficulty, in the following, we shall derive an approximate solution (a lower bound) based on the so-called *link independence assumption*: the link *overload* events (i.e., exceeding the target utilization threshold) occur on different links *independently*. Clearly this assumption does not hold in reality, but it enables us to express B_r in terms of $B_l(\rho_l, c_l)$, the probability that the target utilization level η_l on link l is exceeded, where $\rho_l = \sum_{r:l \in r} \rho_r$. (Again, we may drop the variables ρ_l and c_l in $B_l(\rho_l, c_l)$ if there is no confusion.) Such link independence

assumption has been used extensively in teletraffic analysis and capacity planning in the telephone networks (see e.g., [48]). Under the link independence assumption, the service QoS violation probability B_r, i.e., at least one of the links on route r is overloaded, is given by

$$B_r = 1 - \prod_{l \in r}(1 - B_l). \tag{8.7}$$

Before we present the approximate optimal solution, we need to introduce one more set of notations. Define a small real number $\delta > 0$. For each route r, let $\hat{\rho}_r > \bar{\rho}_r$ be such that

$$\int_{\hat{\rho}_r}^{\infty} \rho_r d\rho_r \leq \delta. \tag{8.8}$$

Since $\int_{\hat{\rho}_r}^{\infty} \rho_r d\rho_r \geq \hat{\rho}_r \int_{\hat{\rho}_r}^{\infty} d\rho_r = \hat{\rho}_r Pr\{\rho_r \geq \hat{\rho}_r\}$, we have $Pr\{\rho_r \geq \hat{\rho}_r\} \leq \delta/\hat{\rho}_r$. In other words, (8.8) basically says that $\hat{\rho}_r$ is such that the probability the traffic demand along route r exceeds $\hat{\rho}_r$ is very small, and thus negligible.

With these notations in place, we now present a lower bound on $E(W)$ as follows (see Appendix D.1 for the detailed derivation).

$$E(W) \geq \sum_{r \in R} e_r \bar{\rho}_r - \sum_{l \in L} \Phi(c_l) - \sum_{r \in R} \pi_r \bar{\rho}_r B_r(\{\hat{\rho}_r\}) - \sum_{r \in R} \pi_r \delta(1 + \sum_{r' \neq r} \frac{\bar{\rho}_r}{\hat{\rho}_{r'}}), \tag{8.9}$$

Denote the right-hand side of the above equation by V, then $E(W) \geq V$. Comparing the lower bound V with the expected net income $\bar{W} = \sum_{r \in R} e_r \bar{\rho}_r - \sum_{l \in L} \Phi_l(c_l)$ *without taking penalty into account*, we see that ignoring the extremal traffic demands (i.e., when $\rho_r \geq \hat{\rho}_r$), we pay at most a penalty of $\pi_r B_r(\{\hat{\rho}_r\})$ per unit of traffic demand on route r for potential service QoS violations. For given $\delta > 0$, the penalty incurred due to extremal traffic demands is upper bounded by $\sum_{r \in R} \pi_r \delta(1 + \sum_{r' \neq r} \frac{\bar{\rho}_r}{\hat{\rho}_{r'}})$. Note also that $B_r(\{\hat{\rho}_r\})$ is the probability of service QoS violation along route r when the long-term average traffic demands are assumed to be $\hat{\rho}_r$. Thus in using V as an approximation to $E(W)$, we are being conservative by over-estimating the probability of potential QoS violations.

From $E(W) \geq V$, we have $\max_{\{\epsilon_r\}} E(W) \geq \max_{\{\epsilon_r\}} V$. Therefore we can obtain the *best* overprovisioning parameters that maximize V instead of the expected net income $E(W)$ as an approximate solution to the original optimization problem (8.6). Using the solution to the basic bandwidth provisioning problem (8.2), we assume $c_l = (1 + \epsilon_l)\bar{\rho}_l/\eta_l$ for a

given set of $\{\epsilon_l\}$, i.e., the target utilization constraints (8.1) hold with equality. Under this assumption, let $\{\epsilon_l^*\}$ be the solution to the optimization problem $\max_{\{\epsilon_r\}} V$, and refer to them as the *approximate optimal overprovisioning parameters*. In the following we demonstrate how $\{\epsilon_l^*\}$ can be derived.

Using (8.7) we can re-write V as follows:

$$V = \sum_{r\in R}(e_r - \pi_r)\bar{\rho}_r - \sum_{l\in L}\Phi_l(c_l) + \sum_{r\in R}\pi_r\bar{\rho}_r\prod_{l\in r}(1-B_l(\hat{\rho}_l, c_l)) - \sum_{r\in R}\pi_r\delta(1+\sum_{r'\neq r}\frac{\bar{\rho}_r}{\hat{\rho}_{r'}}), \quad (8.10)$$

where $\hat{\rho}_l = \sum_{r:l\in r}\hat{\rho}_r$.

Assume B_l is a continuous and everywhere differentiable function of c_l. (See the next section for a discrete case.) For each link l, define

$$\hat{s}_l = \sum_{r:l\in r}\pi_r\bar{\rho}_r\prod_{k\in r, k\neq l}[1 - B_k(\hat{\rho}_k, c_k)]\zeta_l, \quad (8.11)$$

where $\zeta_l = -\frac{d}{dc_l}B_l(\hat{\rho}_l, c_l)$.

Through some simple algebraic manipulation, it is not too hard to show that

$$\frac{\partial V}{\partial \epsilon_l} = \frac{\partial V}{\partial c_l}\frac{\partial c_l}{\partial \epsilon_l} = (-\frac{\partial \Phi_l(c_l)}{\partial c_l} + \hat{s}_l)\frac{\bar{\rho}_l}{\eta_l}. \quad (8.12)$$

Suppose that $\{\epsilon_l^*\}$ are strictly positive, then a necessary condition for them to be an optimal solution is that the gradient ∇V (with respect to $\{\epsilon_l\}$) must vanish at ϵ_l^*'s. Thus from (8.12) we must have

$$\frac{\partial \Phi_l(c_l)}{\partial c_l} = \hat{s}_l \ \ \forall l \in L. \quad (8.13)$$

Intuitively, $\hat{s}_l$ measures the sensitivity of potential penalty reduction to bandwidth increase on link l, whereas $\frac{\partial \Phi_l(c_l)}{\partial c_l}$ measures the sensitivity of bandwidth cost to bandwidth increase on link l. Hence the "optimal" (or rather, the approximate optimal) overprovisioning parameter ϵ_l^* should be chosen such that the two values coincide. In the following discussion, we will loosely refer to $\hat{s}_l$ as the "per-unit bandwidth gain in potential penalty reduction" and $\frac{\partial \Phi_l(c_l)}{\partial c_l}$ as the "increase in per-unit bandwidth cost."

In the above derivation of the approximate optimal solution to the static bandwidth provisioning problem, we have simply assumed the existence of B_l, the probability that the target utilization level η_l on link l is exceeded. The particular form of it depends on the distribution of (average) traffic demands on the link. In the following sections, we consider two different traffic demand models—a self-similar traffic demand model and a traffic demand model based on real Internet traffic measurements to demonstrate the static bandwidth provisioning problem.

8.4.1 $M/G/\infty$ Traffic Demand Model

Since the pioneering work by Leland, Taqqu, Willinger and Wilson [53], the self-similar (or long-range dependent) property has been observed in Ethernet Local Area Network [53], Wide Area Network [63], and World Wide Web traffic [19]. The observed self-similar property of the Internet traffic has important implications on the dementioning and provisioning of the IP networks. In this section, we consider a traffic demand model, $M/G/\infty$, that captures the (asymptotically) self-similar property of the Internet traffic [61, 63].

Consider an $M/G/\infty$ queue, where the service time has a heavy-tailed distribution. We assume that the distribution of the service time has a finite mean. Let X_t denote the number of customers in the system at time t, for $t = 0, 1, 2, \ldots$. Then the count process $\{X_t\}_{t=0,1,2,\ldots}$ is asympotically self-similar. Let ρ denote the customer arrival rate to the $M/G/\infty$ queue and μ the mean service time, then X_t has a Poisson marginal distribution with mean $\rho\mu$ [18].

Now we are ready to present the $M/G/\infty$ traffic demand model on each route. Consider an arbitrary route r. We assume that the traffic demand (i.e., the average traffic arrival rate per unit time) is governed by the count process $\{X_t\}_{t=0,1,2,\ldots}$ of an $M/G/\infty$ queue. Let ρ_r denote the mean traffic demand on the route. It is easy to see that $\rho_r = \rho\mu$, where ρ and μ are the customer arrival rate and the mean service time, respectively, of the $M/G/\infty$ queue. As traffic demands along all the routes are assumed to be independent, the average overall traffic load on a link l is $\rho_l = \sum_{r:l\in r} \rho_r$.

Given the average overall load ρ_l and the link capacity c_l, it can be shown that the probability that the total load on link l exceeds $\bar{c}_l = \eta_l c_l$ during any given unit of time is given by $B_l(\rho_l, c_l) = (\sum_{i=(\bar{c}_l+1)}^{\infty} \frac{\rho_l^i}{i!})e^{-\rho_l}$. Extending the definition of $B_l(\rho_l, c_l)$ to non-integer values of c_l by linear interpolation. At integer values of c_l define the derivative of $B_l(\rho_l, c_l)$ with respect to c_l to be the left derivative. Then $\frac{d}{dc_l}B_l(\rho_l, c_l) = B_l(\rho_l, c_l) - B_l(\rho_l, c_l - 1)$. Therefore, $\zeta_l = -\frac{d}{dc_l}B_l(\hat{\rho}_l, c_l) = \eta_l\{B_l(\hat{\rho}_l, (\eta_l c_l - 1)) - B_l(\hat{\rho}_l, \eta_l c_l)\} = \eta_l \frac{\hat{\rho}_l^{\lceil \eta_l c_l \rceil}}{\lceil \eta_l c_l \rceil !} e^{-\hat{\rho}_l}$. By

this definition of B_l, we are able to obtain the (approximate) optimal overprovisioning parameters ϵ_l^*'s by solving (8.13).

We now discuss the *shapes* of $\hat{s}_l$' and Φ_l on (approximate) optimal overprovisioning parameters ϵ_l^*'s as well as their implication in static bandwidth provisioning. Note first that the shape of $\hat{s}_l$ is determined by ζ_l, which has a shape of (skewed) bell-shape with a center approximately at $\hat{\rho}_l$ (it is essentially a Poisson probability density function). Hence $\hat{s}_l$ is a concave function of $\epsilon_l \geq 0$. In particular, there exists $\hat{\epsilon}_l$ such that $\hat{s}_l$ is an increasing function in the range $[0, \hat{\epsilon}_l]$ and a decreasing function in the range $[\hat{\epsilon}_l, \infty)$ (see Figure 8.3). Intuitively, this means that as ϵ_l moves from 0 towards $\hat{\epsilon}_l$, there is an increasing benefit in bandwidth overprovisioning in terms of *reducing potential QoS violation penalty*. However, as ϵ_l moves beyond $\hat{\epsilon}_l$, there is a *diminished return* in overprovisioning in terms of reducing potential QoS violation penalty.

Suppose that Φ_l' is a linear function, i.e., $\Phi_l(c_l) = \phi_l c_l$. Then $\frac{\partial \Phi_l(c_l)}{\partial c_l} = \phi_l$. Hence (8.13) becomes $\phi_l = \hat{s}_l$. Suppose $\phi_l = \hat{s}_l$ holds for some $\epsilon_l \geq 0$. Because of the shape of $\hat{s}_l$, there potentially exists two solutions $\epsilon_{l,1}$ and $\epsilon_{l,2}$, $0 \leq \epsilon_{l,1} \leq \hat{\epsilon}_l \leq \epsilon_{l,2}$ such that $\phi_l = \hat{s}_l$. In particular, as $\hat{s}_l$ is a decreasing function in the range $[\hat{\epsilon}_l, \infty)$, $\epsilon_{l,2}$ always exists. As $\frac{\partial V}{\partial c_l}$ is positive in the range $(\epsilon_{l,1}, \epsilon_{l,2})$, and is negative in the ranges $[0, \epsilon_{l,1})$ and $(\epsilon_{l,2}, \infty)$, we see that with respect to link l, V is maximized at either $\epsilon_l^* = \epsilon_{l,2}$ or at $\epsilon_l^* = 0$ (whereas it is minimized at $\epsilon_{l,1}$). Intuitively, when only a small amount of bandwidth is overprovisioned on link l, the per-unit bandwidth gain in potential penalty reduction is too small to offset the per-unit bandwidth cost, hence V decreases. However, as we increases the amount of bandwidth overprovisioned, the per-unit bandwidth gain in potential penalty reduction becomes sufficiently large and offsets the per-unit bandwidth cost, hence V increases until it reaches a maximum. Due to the diminished return in the per-unit bandwidth gain in potential penalty reduction, V decreases again when too much bandwidth is overprovisioned on link c. In the special case that ϕ_l is such that $\phi_l > \hat{s}_l$ for all $\epsilon_l \geq 0$, then as $\frac{\partial V}{\partial c_l} < 0$, V attains its maximum at $\epsilon_l^* = 0$ with respect to link l. Intuitively it says that when the per-unit bandwidth cost on link l is higher than the per-unit bandwidth gain in potential penalty reduction, there is no benefit in overprovisioning any bandwidth on link l to guide against any potential QoS violation penalty. These observations can be extended to other bandwidth cost functions such as concave or convex cost functions. In general we see that the trade-off between the bandwidth cost and overprovisioning bandwidth to guide against service QoS violations is critical to the problem of SON bandwidth provisioning. It is also clear from the above discussion that as the per-unit bandwidth cost decreases, there is more

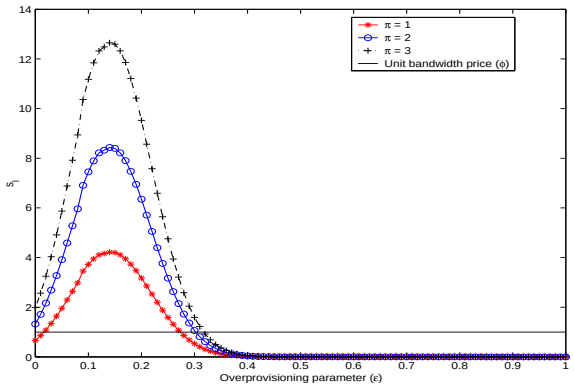


Figure 8.3: Relationship between $\hat{s}_l$, ϵ, & ϕ_l.

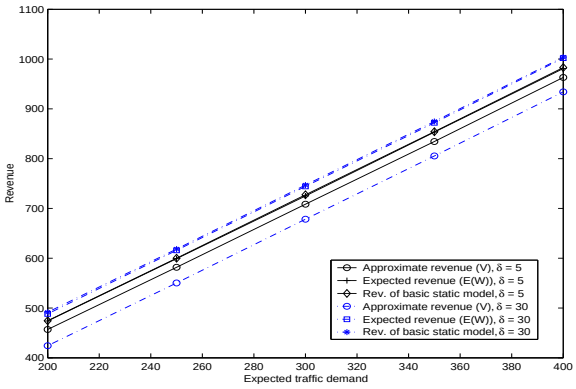


Figure 8.4: Comparison of V and $E(W)$.

benefit in overprovisioning. Lastly, we comment that from (8.13) and (8.11) and the above observations, we can compute the approximate optimal overprovisioning parameters ϵ_l^*'s using *fixed point approximation*.

8.4.1.1 Numerical Examples

We conduct numerical studies to illustrate the properties of the analytic results we obtained and demonstrate the effects of various parameters on static bandwidth provisioning. For this purpose, we consider a simple setting: a single route over a single link. Numerical studies in more complex settings will be performed in a later section.

Unless other stated, the following parameters will be used in the numerical studies: the long-term average traffic demand on the route is 200 (measured in unit of bandwidth per unit of time), i.e., $\bar{\rho}_r (= \bar{\rho}_l) = 200$, and $e_r = 4$, $\phi_l = 1$, $\pi_r = 2$. We set $\delta = 5$ and the target utilization threshold $\eta_l = 0.8$.

Figure 8.3 shows $\hat{s}_l$ as a function of ϵ_l with three different values of π_r: $\pi_r = 1, 2, 3$. In the figure we also include a line corresponding to $\phi_l = 1$ to illustrate how ϵ_l^* can be obtained as the solution to $\hat{s}_l = \phi_l$. Recall from Section 8.4, $\epsilon_l^* = \epsilon_{l,2}$ (the right intersecting point). From Fig. 8.3 we see that as the penalty π_r increases, ϵ_l^* also increases. Hence for higher penalty it is necessary to overprovision more bandwidth to guide against potential QoS violations. Likewise, as we increase the per-unit bandwidth cost ϕ_l (i.e., moving up the line of ϕ_l), ϵ_l^* decreases. In other words, as the bandwidth cost increases, it is beneficial to reduce overprovisioned bandwidth so as to maximize the net income.

In Figure 8.4 we compare the lower bound V with the actual expected net income $E(W)$ for

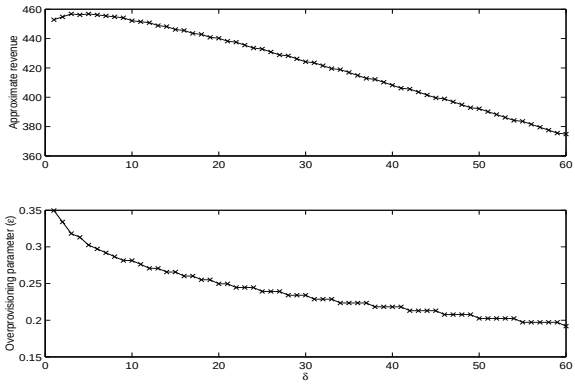


Figure 8.5: Impact of δ on V and ϵ^*.

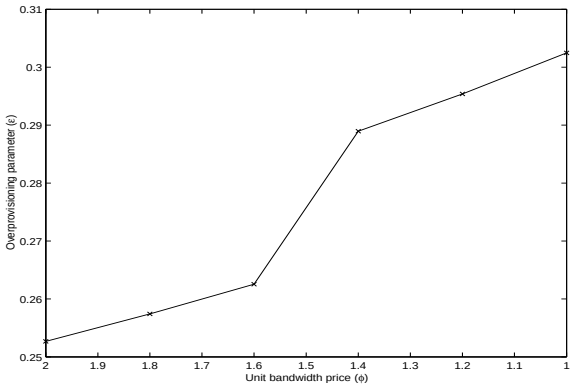


Figure 8.6: Impact of unit bandwidth price on ϵ^*.

two given values of δ (5 and 30). For comparison, we also include the expected net income $\bar{W}$ under the basic static model, where the overprovisioning parameter ϵ_l^* is obtained from the static-penalty model. From the figure we see that for both values of δ, the lower bound V provides a reasonable approximation to $E(W)$. Note also that the difference between the actual expected net income $E(W)$ under the static-penalty model and the expected net income $\bar{W}$ under the basic static is almost invisible. This is likely due to the fact that the additional revenue generated when the traffic demand exceeds its long-term average (the first term in $E(W)$) and the potential penalty incurred due to service QoS violations (the third term in $E(W)$) cancel each other out on average. From Fig. 8.4 it is clear that the lower bound depends on the choice of δ. The smaller the δ is, the closer the approximate revenue V is to the expected revenue $E(W)$. To further explore the relation between δ and V, in Fig. 8.5 we plot V as a function of δ (upper plot). In the figure, we also include the overprovisioning parameter ϵ_l^* as a function of δ (lower plot). We see that V is a concave function of δ, and thus there is a unique δ that maximizes V. On the other hand, ϵ_l^* is a non-increasing function of δ.

To highlight the relationship between bandwidth cost and overprovisioning in Fig. 8.6 we plot the overprovisioning parameter ϵ_l^* as a function of the per-unit bandwidth cost ϕ_l. We see that as the per-unit bandwidth cost ϕ_l decreases (from 2 to 1), the overprovisioning parameter ϵ_l^* increases, i.e., it is more beneficial to overprovision more bandwidth. This is not surprising.

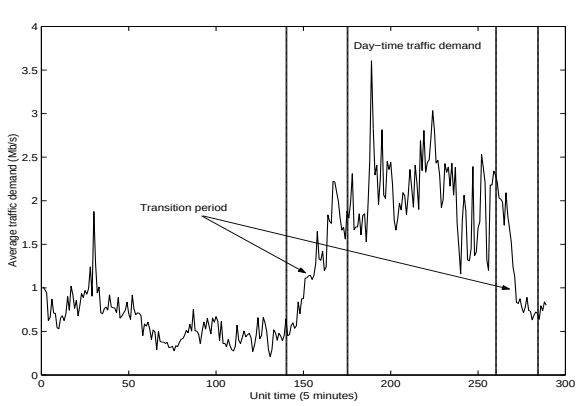


Figure 8.7: Traffic demands of the Auckland data trace.

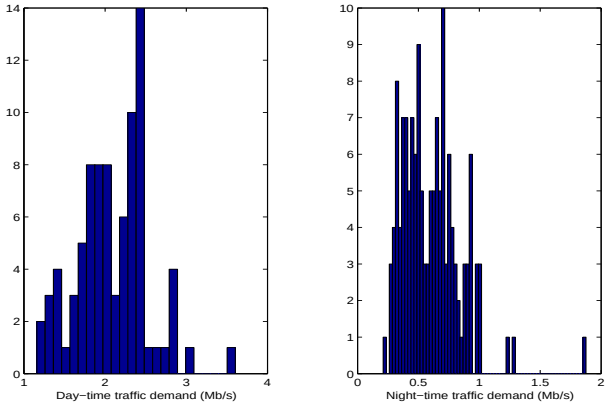


Figure 8.8: Histogram of the Auckland data trace's traffic demands.

8.4.2 Measurement-Based Traffic Demand Model

A key property of the presented approximate optimal solution to the static bandwidth provisioning problem is that it only relies on the marginal distribution of the traffic demand on each link. In this section, we will study the static bandwith provisioning problem based on the measurements of real Internet traffic. That is, we estimate the marginal distributions of the traffic demands on the links by the long-term measurements of the traffic data, and then apply the estimated marginal distributions of the traffic demands to our static bandwidth provisioning problem.

The data trace we will use was collected at the University of Auckland Internet access link on December 1, 1999, lasted roughly for 24 hours (refered to as *Auckland data trace*) [78]. In the Auckland data trace, there are totally 32628004 packet arrivals. Figure 8.7 presents the average traffic arrival rates (i.e. traffic demands) of the Auckland data trace, where each point represents the average traffic demand for a 5 minute time interval (which is also used as the base unit of time, i.e., $t = 5\,minutes$, see Figure 8.2). Given the largely different traffic arrival patterns during the day-time and night-time, we will accordingly provision bandwidth differently for them, where the day-time is defined to be from $10{:}00$AM to $5{:}00$PM and night-time from $7{:}00$PM to $7{:}00$AM. We will refer to the traffic demands during the day-time and night-time as *day-time traffic demand* and *night-time traffic demand*, respectively. All other times are considered to be transition times. The bandwidth provisioned for the day-time and night-time should be switched during the transition times based on certain criteria, which is not considered in this paper.

Now let us consider the properties of the day-time traffic demands and the night-time traffic

Table 8.1: Provisioning for the Auckland traffic demands.

	Mean	STD	C.O.V.	ϵ_l^*	V
Day-time	2096	442	0.21	0.67	3446
Night-time	609	240	0.39	0	1672

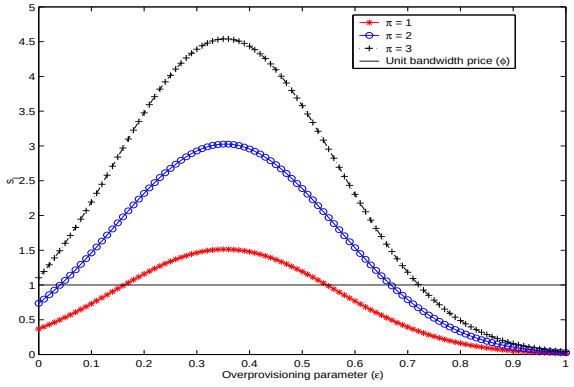


Figure 8.9: Relationship between $\hat{s}_l$, ϵ, & ϕ_l for *Day-time traffic*.

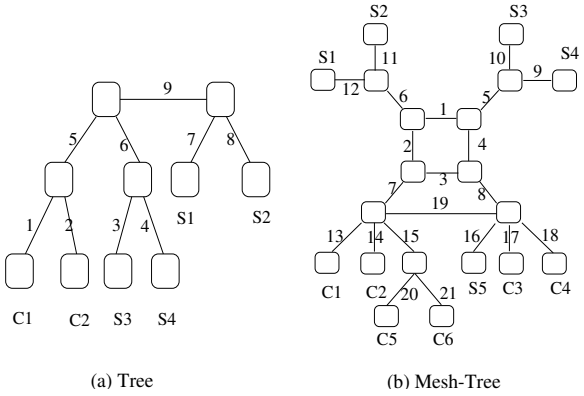


Figure 8.10: SON topologies.

demands. The mean traffic arrival rate over the whole day-time duration is $2.1\,Mb/s$, while over the night-time duration it is $0.6\,Mb/s$. Figure 8.8 plots the histograms of the traffic demands for the day-time (left-hand side) and night-time (right-hand side) separately, where the bin sizes for the day-time traffic demands and the night-time traffic demands are $100\,Kb/s$ and $50\,Kb/s$, respectively. From the plots we see that the day-time traffic demands are relatively symmetrically centered at its mean arrival rate, while the night-time traffic demands are more skewed. In th following studies, we will model the day-time traffic demands by a *Normal* distribution, while the night-time traffic demands by a *Lognormal* distribution to retain the different traffic characteristics during the day-time and night-time. Table 8.1 presents the mean traffic demands and the standard deviations (*STD*) of the day-time and night-time traffic demands, where the base unit of bandwidth (traffic demand) is $1\,Kb/s$.

In the following, we will conduct numerical studies to illustrate the static bandwidth provisioning using the Auckland data trace. In all these studies, we again consider the simple setting: a single route over a single link. The per-unit bandwidth per-unit time earning $e_r = 4$, and $\phi_l = 1$, $\pi_r = 2$. We set the target utilization threshold $\eta_l = 0.8$.

Similar to the numerical example for the $M/G/\infty$ traffic demand model, in Figure 8.9,

we show $\hat{s}_l$ as a function of ϵ_l with three different values of π_r: $\pi_r = 1, 2, 3$, for the day-time traffic demands. The value of δ used is 140. In the figure we also include a line corresponding to $\phi_l = 1$ to illustrate how ϵ_l^* can be obtained as the solution to $\hat{s}_l = \phi_l$ (see (8.13)). Following a similar argument as that in Section 8.4.1, there potentially exists two solutions $\epsilon_{l,1}$ and $\epsilon_{l,2}$, $0 \leq \epsilon_{l,1} \leq \epsilon_{l,2}$ such that $\phi_l = \hat{s}_l$. Moreover, with respect to link l, V is maximized at either $\epsilon_l^* = \epsilon_{l,2}$ or at $\epsilon_l^* = 0$. From Fig. 8.9 we can draw the similar conclutions as that in the $M/G/\infty$ traffic demand model. In particular, we see that as the penalty π_r increases, ϵ_l^* also increases. Hence for higher penalty it is necessary to overprovision more bandwidth to guide against potential QoS violations. Likewise, as we increase the per-unit bandwidth cost ϕ_l (i.e., moving up the line of ϕ_l), ϵ_l^* decreases. In other words, as the bandwidth cost increases, it is beneficial to reduce overprovisioned bandwidth so as to maximize the net income. However, compared with the result in Figure 8.3, we see that we obtain larger overprovisioning parameters here. This is caused by the high traffic fluctuation in the Auckland data trace. Table 8.1 gives the *coefficient of variance* for the day-time traffic demands (and the night-time traffic demands) in the column marked as *C.O.V.*. This value (0.21) is much higher than that in Figure 8.3, which is 0.07.

To compare the different provisioning behaviors during the day-time and night-time, we present the overprovisioning parameters for both the day-time and night-time traffic demands in Table 8.1. To obtain these results, we have searched for the best δ's that yeild the maximal V's, respectively. In the table we also include the approximate revenue V's (per-unit time) for the day-time and night-time traffic demands. From the table we see that for the day-time traffic demands the overprovisioning parameter $\epsilon_l^* = 0.67$, while for the night-time traffic demands $\epsilon_l^* = 0$. The reason is as follows. Even though the average traffic demands during night-time are much lower than that during day-time, we observe a much higher traffic demand fluctuation during the night-time than that during the day-time (see Table 8.1 for their corresponding coefficients of variance). It is too expensive to accommodate this high traffic demand variance during the night-time ($\epsilon_{l,2}$ is dramatically large), therefore, no overprovisioning is provided in this case. During day-time, the (per-unit time) approximate revenue is 3446, which is higher than that during the night-time (1672). This is not unexpected.

8.4.3 Performance Evaluation

We now use two SON topologies—the *tree* (Fig. 8.4.2(a)) and the mesh-tree (Fig. 8.4.2(b)) topologies—to illustrate the effect of traffic load distribution among various routes of a

SON on static bandwidth provisioning. In the following $a \rightarrow b$ denotes a route from service gateway a to service gateway b. The path with *minimum* "hop-count" (i.e., service gateways) is used as the route between two service gateways. In case there are two such paths, only one is chosen. In the numerical studies below, we will use the $M/G/\infty$ traffic demand model. We set $e_r = 10$, $\pi_r = 2$ for all the routes, and $\phi_l = 1$ for all the links. The value of δ is chosen in such a way that $\delta_r = \frac{1}{40}\rho_r$.

In the tree topology, four routes are used: $R1 = S3 \rightarrow C1$, $R2 = S1 \rightarrow C1$, $R3 = S4 \rightarrow C2$, and $R4 = S2 \rightarrow C2$. To investigate the effects of different traffic loads on bandwidth provisioning, we consider two types of traffic load distribution among the routes: the *balanced* load where the expected traffic demand for all routes is 200, and the *unbalanced* load where the expected traffic demands on routes $R1, R2, R3, R4$ are $300, 100, 250$, and 150, respectively. Table 8.2 presents the resulting overprovisioning parameter ϵ_l^* and provisioned bandwidth c_l for six representative links: link $1, 4, 5, 7, 8$, and 9. The corresponding average traffic loads $\bar{\rho}_l$'s on these four links are also given in the table. From the results we see that under the balanced load, links with a higher average traffic load have a smaller overprovisioning parameter. This is due to statistical multiplexing gains for carrying a higher load on a link. In the Unbalanced case, similar results can be observed. Note that even links 4 and 9 have the same traffic demand load, they are overprovisioned differently. This is caused by the fact that, there are two routes traversing link 9 while there is only one on link 4.

Table 8.2: Tree Topology.

Link ID		1	4	5	7	8	9
Balanced	ρ_l	400	200	800	200	200	400
	ϵ_l^*	0.26	0.3	0.23	0.3	0.3	0.26
	c_l	630	325	1230	325	325	630
Unbalanced	ρ_l	400	250	800	100	150	250
	ϵ_l^*	0.26	0.27	0.23	0.41	0.34	0.33
	c_l	630	397	1230	176	251	416

We now consider the mesh-tree topology. In this case there are 10 routes: $R1 = S1 \rightarrow C1$, $R2 = S2 \rightarrow C2$, $R3 = S3 \rightarrow C1$ (1), $R4 = S4 \rightarrow C2$ (1), $R5 = S1 \rightarrow C3$ (3), $R6 = S2 \rightarrow C4$ (3), $R7 = S3 \rightarrow C3$, $R8 = S4 \rightarrow C4$, $R9 = S5 \rightarrow C5$, $R10 = S5 \rightarrow C6$. The number in the parentheses following a route shows a link that the route traverses in case there are multiple paths between the source and destination with the same

Table 8.3: Mesh-Tree Topology.

Link ID		2	6	11	18	19	21
Balanced	ρ_l	1200	800	400	400	400	200
	ϵ_l^*	0.22	0.23	0.26	0.26	0.26	0.3
	c_l	1830	1230	630	630	630	325
Unbalanced	ρ_l	1350	1100	500	400	400	100
	ϵ_l^*	0.21	0.2	0.24	0.26	0.26	0.41
	c_l	2042	1650	775	630	630	176

path length. Again for the balanced load case, all the routes have an average traffic demand of 200; while for the unbalanced load case, the average demands for routes $R1$ to $R10$ are $300, 250, 100, 150, 300, 250, 100, 150, 300$, and 100 respectively. Table 8.3 shows the results for six representative links: link $2, 6, 11, 18, 19$, and 21. From the table we can see that similar observations also hold for the mesh-tree topology.

In this section, we have studied the static bandwidth provisioning mode, where during a relatively long period, the provisioned bandwidth on a link will not be changed. The static bandwidth provisioning mode is simple in bandwidth management, but may result in inefficient bandwidth usage facing traffic demand fluctuations. In the next section, we will study the dynamic bandwidth provisioning mode, where the link bandwidth could be dynamically adjusted according to the traffic demand fluctutions in relatively shorter time intervals.

8.5 Dynamic Bandwidth Provisioning

In this section we study the dynamic bandwidth provisioning problem. As pointed out in Section 8.2, to account for the potential higher cost in supporting dynamic bandwidth provisioning, it is likely that the underlying network domains will charge the SON different prices for statically provisioned and dynamically allocated bandwidth. Hence we assume that for each link l, the cost for reserving c_l amount of bandwidth *statically* is, as before, $\Phi_l(c_l)$; while the cost of reserving the same amount of bandwidth *dynamically* is $\Phi'_l(c_l)$, where $\Phi'_l(c_l) \geq \Phi_l(c_l)$. Given this price differential, *a key question for the SON is to determine how much bandwidth should be reserved statically on each link* l a priori *to meet certain base traffic demands, while dynamically allocating bandwidth to meet the additional traffic demands as needed*. The objective is again to maximize the overall long-

term expected net income of the SON.

To focus on the dynamic bandwidth problem, we assume that the underlying network domains possess abundant bandwidth that the dynamic requests for additional bandwidth from the SON are always satisfied. In other words, no request is blocked. Under this assumption, for a given traffic demand matrix $\{\rho_r\}$, it is possible to compute the expected additional bandwidth that needs to be dynamically allocated to meet the traffic demands. This can be done, for example, using the $M/G/\infty$ traffic demand model introduced in the previous section. However such precise formulation is extremely complicated, and consequently the corresponding optimization problem is unlikely to be tractable. In the following, we will first describe an approximate model based on the marginal distributions of the traffic demands on the links of the overlay network; and then present an adaptive heuristic algorithm for dynamic bandwidth provisioning based on *online* traffic measurements.

8.5.1 Approximate Model

Suppose for each link $l \in L$, c_l amount of bandwidth has been provisioned statically *a priori*. Given a traffic demand matrix $\{\rho_r\}$, we approximate the *expected* additional bandwidth that must be dynamically reserved to meet the traffic demands by the following expression:

$$\Delta c_l = \left\{ \frac{\rho_l}{\eta_l} - c_l \right\}^+, \tag{8.14}$$

where $\rho_l = \sum_{l \in r} \rho_r$. Then $\Delta c_l > 0$ if and only if $\rho_l > \eta_l c_l$.

Using (8.14) we can write down the *approximate* overall net income the SON generates for the given traffic demand matrix $\{\rho_r\}$:

$$\tilde{W}(\{\rho_r\}) = \sum_{r \in R} e_r \rho_r - \sum_{l \in L} \Phi_l(c_l) - \sum_{l \in L} \Phi'_l(\Delta c_l). \tag{8.15}$$

Integrating on both sides of (8.15) over the (joint) distribution of $d\{\rho_r\}$, we have

$$E(\tilde{W}) = \sum_{r \in R} e_r \bar{\rho}_r - \sum_{l \in L} \Phi_l(c_l) - \sum_{l \in L} \int \cdot \int \Phi'_l(\Delta c_l) d\{\rho_r\}. \tag{8.16}$$

The dynamic bandwidth provisioning problem can now be formulated as the following optimization problem:

$$\max_{\{c_l\}} E(\tilde{W}). \tag{8.17}$$

Note that unlike the static bandwidth provisioning problem, here we do not have any explicit QoS or target utilization constraints. This is because we implicitly assume that whenever the target utilization threshold is about to be exceeded, additional bandwidth is dynamically allocated on the link to meet the service QoS. We will refer to the optimization problem (8.17) as the *approximate model* for dynamic bandwidth provisioning. In the following, we will present an (approximate) solution to the approximate model of the dynamic bandwidth provisioning problem. For the detailed analysis, we refer interested readers to Appendix D.2.

Assume both bandwidth cost functions are linear, i.e., for any $l \in L$, $\Phi_l(c_l) = \phi_l c_l$ and $\Phi'_l(\Delta c_l) = \phi'_l \Delta c_l$, where $\phi_l \leq \phi'_l$ for any l. Let c'_l be such that $Pr\{\rho_l > \eta_l c'_l\} = \phi_l/\phi'_l$. Then the set of c'_l's is an (approximate) solution to the dynamic bandwidth provisioning problem, i.e., c'_l is the amount of bandwidth to be statically provisioned, while the portion to be dynamically allocated on link l is given by (8.14), for a given traffic demand matrix $\{\rho_r\}$.

An intuitive interpretation of the above results is that under the dynamic bandwidth allocation model, we need to statically reserve at most c'_l amount of bandwidth on each link l, where the probability that the (average) aggregate load on link l exceeds the statically reserved link bandwidth c'_l equals the ratio of the two prices on the link l, ϕ_l/ϕ'_l. In the special case that $\phi_l = \phi'_l$, i.e., the unit price of dynamically allocated bandwidth is the same as that of the statically reserved one, we have $c'_l = 0$. Hence in this case, no static capacity needs to be reserved.

8.5.1.1 Numerical Examples

In this section we perform numerical studies to illustrate the properties of the dynamic bandwidth provisioning model, and compare it with the static bandwidth provisioning model. Unless otherwise stated, the per-unit bandwidth per-unit time earning $e_r = 4$, and $\phi_l = 1$, $\phi'_l = 1.5$. The target link utilization threshold η_l is 0.8.

In the first set of studies, we examine the effects of the per-unit bandwidth price ϕ'_l for dynamically allocated bandwidth on the amount of bandwidth provisioned statically ***a priori*** c_l and the approximate revenue $E(\tilde{W})$. In these studies, we use the simple network setting: a single route over a single link. The traffic demand model is $M/G/\infty$ and the long term average traffic demand on the route is 200. Figure 8.11 presents the bandwidth provisioned statically c_l (upper plot) and the approximate revenue $E(\tilde{W})$ (lower plot) as functions of ϕ'_l,

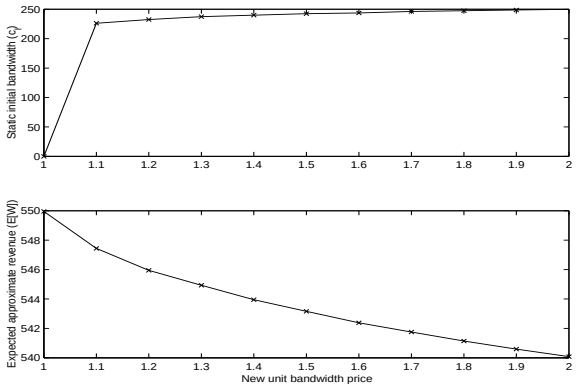


Figure 8.11: Effects of ϕ_l' on c_l and $E(\tilde{W})$.

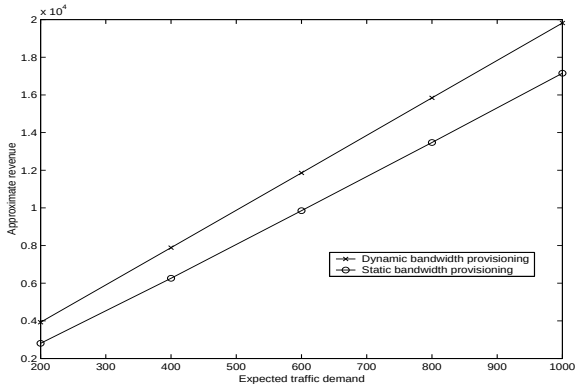


Figure 8.12: Dynamic *vs.* static bandwidth provisioning.

respectively. From the figure we see that as the per-unit bandwidth price for dynamically allocated bandwidth increases, more bandwidth needs to be provisioned statically *a priori*. However, the increase in the amount of static bandwidth is not dramatic as ϕ_l' increases from $\phi_l' = 1.1$ to $\phi_l' = 2$. On the other hand, as we increase the price for dynamically allocated bandwidth, the approximate revenue $E(\tilde{W})$ decreases. This is partly due to the fact that a SON needs to statically provision more bandwidth *a priori* on each link, besides the fact that the SON needs to pay more for the dynamically allocated bandwidth.

In the next set of numerical studies, we compare the dynamic bandwidth provisioning model with the static bandwidth provisioning model in terms of the approximate revenues obtained, using the *tree* network topology (see Figure 8.4.2(a)), with a similar setting as that in Section 8.4.3. In particular, we use the *balanced* traffic load model and assume the traffic demand on each route is governed by the $M/G/\infty$ model. For static bandwidth provisioning, $\pi_r = 2$. Figure 8.12 presents the approximate revenue as a function of the (long-term) average traffic demands for dynamic and static bandwidth provisioning, respectively. From the figure we see that, for both dynamic and static bandwidth provisioning models, the approximate revenue increases as the average traffic demand increases, and the dynamic bandwidth provisioning has a higher approximate revenue than that of the static bandwidth provisioning. Moreover, as the average traffic demand increases, the difference between the approximate revenues of the dynamic bandwidth provisioning and the staic bandwidth provisioning becomes larger. This is possibly due to the fact that, as the average traffic demand on a route increases, traffic along the route becomes more bursty (recall that the marginal distribution of traffic demand on a route is Poisson), and the dynamic bandwidth provisioning model works better than the static bandwidth provisioning in this case.

8.5.2 Adaptive Online Bandwidth Provisioning Algorithm

In developing the approximate dynamic bandwidth provisioning model, we have assumed that the (average) traffic demands are known *a priori* for determining the additional bandwidth that must be dynmaically allocated to meet the traffic demands (see (8.14)). In this section, we present an *adaptive online* bandwidth provisioning model (or simply online dynamic model) that dynamically adjust the allocated bandwidth on a link according to the *measurement* of the traffic demands on the links of the network.

As before, let $\bar{\rho}_r$ denote the long-term average traffic demand on route r, and $\bar{\rho}_l = \sum_{r:l\in r} \bar{\rho}_r$, the long-term average traffic demand on link l. Based on the measurement of the traffic demands on the links, our target in this section is to determine the amount of bandwidth c_l that should be statically provisioned *a priori* to meet certain base traffic demands, and the amount of bandwidth Δc_l that should be allocated dynamically to accommodate the traffic demand dynamics in the network.

Let t denote a fixed time interval. In the online dynamic model, the average traffic demand ρ_r during each such time interval is calculated at the end of the time interval. Based on the measured average traffic demands and the contracted service QoS, the bandwidth allocated on each link will be adjusted accordingly at the end of the time interval. Moreover, the resulted bandwidth will be kept constant during the next measurement time interval. In other words, the allocated bandwidth is only adjusted at the end of each measurement time interval. To reduce the frequency of allocating additional bandwidth or de-allocating extra bandwidth caused by short-term traffic fluctuations, bandwidth will be allocated in units of quota, which is a chunk of bandwidth [88] and normally much larger than one unit of bandwidth. In the following, we will denote the size of a quota by Θ (in unit of bandwidth).

Let c_l denote the amount of bandwidth that has been provisioned statically *a priori*. In the online dynamic model, c_l is chosen in such a manner that, if the average traffic demand on a link l does not exceed $\bar{\rho}_l$, the service QoS will be honored, i.e.,

$$c_l = \lceil \frac{\bar{\rho}_l}{\eta_l \Theta} \rceil \Theta, \tag{8.18}$$

note that, the initial static bandwidth is allocated in units of quota.

Next, we discuss the allocation of additional bandwidth and de-allocation of extra bandwidth on an arbitrary link l. To reduce the possibility that the service QoS is violated, the

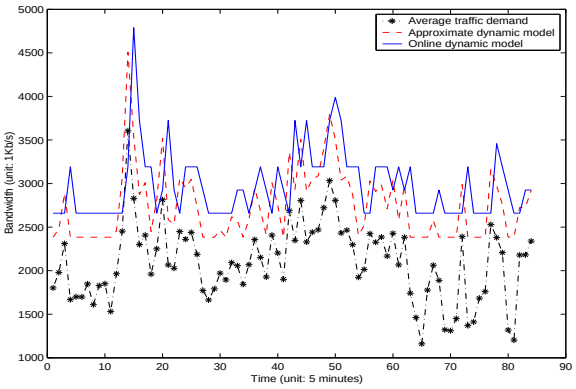


Figure 8.13: Dynamic bandwidth provisioning with approximate model and online model.

online dynamic model will allocate the additional bandwidth (a new quota) when the average traffic demand is approaching the target link utilization level threshold, instead of until the thresold is exceeded. Let ι^f denote a positive number, and C_l the current total bandwidth on link l, i.e., $C_l = c_l + \Delta c_l$. Then an additional quota will be allocated onto link l if $\rho_l > C_l\eta_l - \iota^f$. ι^f is called the forward threshold for allocating a new quota. Similarly, a backward threshold for de-allocating an extra quota is defined as: (denoted by ι^b (a positive number)): an extra quota is released from link l only if $\rho_l < (C_l - \Theta)\eta_l - \iota^b$.

Because the online dynamic model only adjusts bandwidth on the links at the end of the each measurement interval, it is possible that the service QoS is violated during the course of the measurement time interval. As in static bandwidth provisioning with penalty in Section 8.4, certain penalty will apply in this case. Let π_r denote the average penalty suffered by per unit of traffic demand per unit of time (the measurement time interval) along route r *when the service QoS along route* r *is violated*. Then the revenue of the online dynamic model for a measurement time interval is,

$$\bar{V} = \sum_{r\in R} e_r\rho_r - \sum_{l\in L}\Phi_l(c_l) - \sum_{l\in L}\Phi'_l(\Delta c_l) - \sum_{r\in R}\pi_r\rho_r\mathbf{1}_{\{\rho_l/C_l>\eta_l:l\in r\}}, \tag{8.19}$$

where the indicator function $\mathbf{1}_{\{\rho_l/C_l>\eta_l:l\in r\}} = 1$ if $\rho_l/C_l > \eta_l$ holds for any link l on route r, 0 otherwise.

In the following, we perform numerical studies to illustrate the bandwidth allocation behavior of the online dynamic model. The studies are carried out in the simple network setting using the *day-time traffic demands* of the *Auckland data trace* (see Fig. 8.7). The following parameters are used. The base unit of bandwidth for the Auckland data trace is

$1\,Kb/s$. The measurement time interval (i.e., unit time) is $5\,minutes$. The per-unit bandwidth per-unit time earning $e_r = 4$, and $\phi_l = 1$, $\phi'_l = 1.5$, $\pi_r = 2$. The target utilization threshold $\eta_l = 0.8$. The size of quota $\Theta = 0.6\sigma$, where σ is the standard deviation of the day-time traffic demands of the Auckland data trace (see Table 8.1). The forward and backward threshold $\iota^f = \iota^b = 0.3\Theta$.

Figure 8.13 presents the average traffic demands (per $5\,minutes$) and the corresponding provisioned bandwidth in the online dynamic model. For the purpose of comparison, we also include the bandwidth provisioning behavior of the approximate dynamic model. From the figure we see that the online dynamic model is able to adjust the link bandwidth according to the dynamics of the traffic demands on the link and meanwhile remains insensitive to small short-time fluctuations in traffic demands (for example, see the provisioned bandwidth at time $24, 25$ and 26). Because of the nature of the online dynamic model, sometimes the bandwidth on a link could be less than the average traffic demand on the link (for example, at time 14), where a penalty will apply. (A penalty may apply in other cases.) Note also that, under this parameter setting, the approximate dynamic model has a smaller initial static bandwidth than the online dynamic model. Moreover, the approximate dynamic model is more sensitive to the fluctuations in traffic demands than the online dynamic model.

Table 8.4: Per-unit time average revenue.

	Approximate model	Online model
Average revenue	5468	4152

Table 8.4 gives the mean revenues (per-unit time) of the approximate dynamic model and the online dynamic model, averaged over the whole duration of the day-time traffic demands of the Auckland data trace. From the table we see that the approximate dynamic model has a higher per-unit time average revenue than the online dynamic model. There are possibly two reasons. First, under this parameter setting, the amount of initial static bandwidth is larger than the approximate dynamic model, therefore causing more cost on the overlay. Second, the online dynamic model is measurement-based and the bandwidth on a link is only adjusted at the end of the measurement time itnervals. Consequently, as we discussed before, service QoS could be violated during a time interval and incurs penalty on the overlay. However, the online dynamic model has the advantage that it does not make any assumption about the (average) traffic demands (except the long-term average traffic demand and its standard deviation).

8.6 Summary

In this chapter, we studied the bandwidth provisioning problem for the service overlay networks. We considered both the static and dynamic bandwidth provisioning models and our study took into account various factors such as service QoS, traffic demand distributions, and bandwidth costs.

The approximate optimal solution we presented to the static bandwidth provisioning problem is generic in the sense that it applies to different marginal distributions of the traffic demands on the routes in a network, which makes the solution very attractive facing different traffic arrival behaviors. The static bandwidth provisioning model is simple in terms of network resource management but may result in inefficient network resource usage if the traffic demands are highly variable. In this kind of environments, the dynamic bandwidth provisioning model outperforms the static bandwidth provisioning model, albeit with more complex and frequent network resource managements. We investigated the effects of various parameters like static and dynamic bandwidth costs on the revenue that a SON can obtain, which provides useful guidelines on how a SON should be provisioned to stay profitable.

Part IV

Conclusions and Future Work

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This dissertation addressed the scalability issues in supporting QoS from two complementary aspects, namely the packet forwarding data plane and the network resource management control plane. On the packet forwarding data plane, a virtual time framework was proposed as a unifying packet scheduling framework to provide *scalable* support for guaranteed services in the Internet. In this framework, Internet core routers do not need to maintain any per-flow state and do not perform any per-flow operations. Consequently, they are able to handle a large number of simultaneous flows. The key notion in the virtual time framework is a *virtual timestamp*, which is initialized at network edge routers and referred and/or updated by network core routers, depending on the service granularity supported by the network. Several new *core stateless* packet scheduling algorithms were designed and analyzed to illustrate how both aggregate and per-flow service guarantees can be supported within this same framework. This is critical for the Internet to continue its evolution. Moreover, we investigated the cost-performance trade-offs in supporting QoS in the Internet by studying the QoS provisioning power of these packet scheduling algorithms. Figure 9.1 summarizes the packet scheduling algorithms we have studied in this dissertation and their trade-offs in supporting QoS in the Internet. Compared to Figure 1.2 we see that a whole scheduling spectrum, ranging from the current best-effort FIFO network to the most advanced per-flow QoS provisioning scheme (VTRS), were studied.

On the network resource management control plane, two scalable bandwidth broker architectures were designed and investigated. The first one was a centralized bandwidth broker

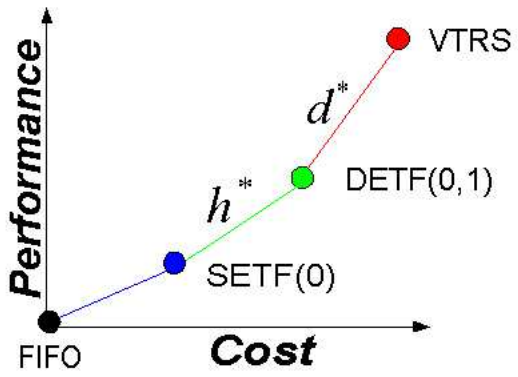


Figure 9.1: Cost-performance trade-offs in supporting QoS in the Internet

architecture, which is built upon the core stateless packet scheduling algorithms we designed. By conducting admission controls on a per-path basis instead of on a "hop-by-hop" basis, this bandwidth broker architecture significantly reduces the complexity of the admission control algorithm; therefore, it improves the scalability of existing bandwidth broker architectures. To further improve its scalability, a hierarchical bandwidth broker architecture was designed. In this architecture, multiple edge bandwidth brokers are deployed in a network, along with the conventional centralized bandwidth broker. Edge bandwidth brokers handle the flow admission control and resource management functionalities for certain pre-defined paths. They interact with the centralized bandwidth broker for allocating and de-allocating trunk bandwidth along the paths. In this way, the centralized bandwidth broker only needs to handle coarser time scale trunk bandwidth requests from edge bandwidth brokers. Consequently, its scalability is greatly improved.

Finally, to provide real end-to-end QoS support and to facilitate the creation and deployment of *value-added services* such as VoIP, Video-on-Demand, and other emerging QoS-sensitive services over the Internet, an architecture called the *service overlay network* (SON) was proposed. Special servers, called service gateways, are deployed at certain strategically selected locations over the Internet to aid the data forwarding and resource management. The bandwidth provisioning problem for a service overlay network was mathematically formulated and investigated, taking into account various factors such as SLA, QoS, traffic demand distributions, and bandwidth costs. Analytical models and approximate solutions were developed for both static and dynamic bandwidth provisioning, which provide useful guidelines on how a SON should be provisioned to stay profitable.

9.2 Future Work

In the following we briefly discuss possible future research directions that we plan to explore.

9.2.1 Packet Forwarding Data Plane

So far, we have primarily focused on providing *deterministic* QoS guarantees in the virtual time framework. The delay bounds reported in this dissertation are the *worst-case* delay bounds. On the other hand, many real-time multimedia applications may be more interested in *average* delay bounds. Therefore, it is of great interest and importance to study and provide *stochastic* QoS guarantees in the Internet.

In the aggregate packet scheduling schemes we have studied, all flows traversing a network will receive the same degree of service guarantees from the network. In reality, it may be desirable to support multiple service classes instead of a single service class. In this way, users can choose different service classes depending on the requirements from different applications.

In establishing the results of the Virtual Time Reference System, we have assumed a fine-grained time granularity. However, we may only have finite bits to encode the packet state in reality. We will study VTRS with a coarser-grained time granularity. Specifically, as in the case of SETF and DETF, we will extend VTRS to a slotted-time environment, which may greatly reduce the cost of VTRS, while still being able to provide per-flow service guarantees.

9.2.2 Network Resource Management Control Plane

In the current bandwidth broker architecture design, we focused on the bandwidth allocation and management problem *within a single network domain*. However, the bandwidth allocation problem can be different in a multiple domain environment, where different network domains may belong to different administrations. Consequently, an end-to-end bandwidth negotiation scheme may be desirable, be it on a per-flow basis or on a per-organization basis.

IP Telephony is a promising service in the future Internet. However, it also imposes certain unique challenges on the management of network resources. For example, it may be necessary to conduct dynamic QoS control and management such as admission control and/or

resource provisioning at the time scale of flow arrival and departure. Therefore, an efficient and scalable bandwidth allocation and management scheme is critical to the performance of the IP networks in support of the IP Telephony service. Some initial results concerning the network resource management for supporting IP Telephony in the Internet were reported in [24]. Currently, we are investigating the feasibility of probabilistic admission control schemes to better utilize network resources.

9.2.3 Service Overlay Networks

The bandwidth provisioning problem for service overlay networks was studied in this dissertation. However, many issues regarding the design and implementation of the SON architecture remain unanswered. For example, how many service gateways are needed to construct a SON to maximize the revenue of a SON operator, while still being able to provide certain degrees of QoS guarantees? Where should these service gateways be placed? Furthermore, how should user traffic be routed within a SON to optimize its resource usages? All of these questions are important for a SON to stay profitable. We plan to investigate these issues in the future work.

9.2.4 Inter-Domain Internet Routing

The Border Gateway Protocol (BGP) is the current *de facto* Internet inter-domain routing protocol that glues the whole Internet together [65]. Theoretically speaking, when the network is stable, we should seldom see routing dynamics. However, significant routing dynamics have been observed on the Internet from time to time [51, 52]. Internet routing instability has an adverse impact on application performance. It manifests itself with increased network latencies and packet losses [52, 62]. It is therefore important to understand the underlying reasons for these routing dynamics, and then design proper apparatus to reduce the degree of routing dynamics. Specifically, we plan to investigate the characteristics of routing dynamics observed at multiple vantage points, and understand the causes of and relationships among the routing dynamics. We would also like to study the path exploration behaviors associated with BGP, or more generally, path-vector routing protocols, to enhance their stability [8, 56, 64]. Some initial results were reported in [23].

Bibliography

[1] D. G. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. 18th ACM SOSP*, Banff, Canada, October 2001.

[2] M. Andrews, B. Awerbuch, A. Fernandez, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proceedings of the 37th Annual Symposium on Foundations Of Computer Science*, pages 380–389, Burlington, VT, October 1996.

[3] P. Aukia, M. Kodialam, P. Koppol, T. Lakshman, H. Sarin, and B. Suter. RATES: A server for MPLS traffic engineering. *IEEE Network*, pages 34–41, March/April 2000.

[4] Y. Bernet, J. Binder, S. Blake, M. Carlson, B. E. Carpenter, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, and W. Weiss. A framework for differentiated services. Internet Draft, February 1999. Work in Progress.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, December 1998.

[6] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: An overview. RFC 1633, June 1994.

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 functional specification. RFC 2205, September 1997.

[8] A. Bremler-Barr, Y. Afek, and S. Schwarz. Improved bgp convergence via ghost flushing. In *Proc. IEEE INFOCOM*, San Francisco, CA, April 2003.

[9] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. A framework for multiprotocol label switching. Internet Draft, September 1999. Work in Progress.

[10] V. Cerf and R. Kahn. A protocol for packet network interconnection. *IEEE Transactions on Communications Technology*, COM-22(5):627–641, May 1976.

[11] A. Charny and J.-Y. Le Boudec. Delay bounds in a network with aggregate scheduling. In *Proceedings of QoFIS*, Berlin, Germany, October 2000.

[12] Y. Chawathe, S. Fink, S. McCanne, and E. Brewer. A proxy architecture for reliable multicast in heterogeneous environments. In *Proceedings of ACM Multimedia*, Bristol, U.K., September 1998.

[13] D. Clark. Adding services discrimination to the Internet. Technical report, MIT, August 1995.

[14] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: architecture and mechanism. In *Proc. ACM SIGCOMM*, August 1992.

[15] K. Coffman and A. Odlyzko. Internet growth: Is there a "Moore's Law" for data traffic? In *Handbook of Massive Data Sets*. Kluwer, 2001.

[16] Virtela Communications. http://www.virtele.com.

[17] Internap Network Services Corporation. http://www.internap.com.

[18] D. Cox and V. Isham. *Point Processes*. Chapman and Hall, 1980.

[19] Mark Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and causes. In *Proc. ACM SIGMETRICS*, pages 160–169, Philadelphia, PA, May 1996.

[20] R. Cruz. A calculus for network delay, Part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[21] R. Cruz. Sced+: Efficient management of quality of service guarantees. In *Proc. IEEE INFOCOM*, San Francesco, CA, March 1998.

[22] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. ACM SIGCOMM*, pages 1–12, Austin, TX, September 1989.

[23] Z. Duan, J. Chandrashekar, J. Krasky, K. Xu, and Z.-L. Zhang. Damping BGP route flaps. Submitted.

[24] Z. Duan and Z.-L. Zhang. A scalable bandwidth management architecture for supporting VoIP applications using bandwidth broker. In *11th IEEE Workshop on Local and Metropolitan Area Networks*, Boulder, CO, March 2001.

[25] Z. Duan, Z.-L. Zhang, and Y. T. Hou. Service overlay networks: SLAs, QoS and bandwidth provisioning. *To appear in* ACM/IEEE Transactions on Networking.

[26] Z. Duan, Z.-L. Zhang, and Y. T. Hou. Bandwidth provisioning for service overlay networks. In *Proceedings of SPIE ITCOM (Scalability and Traffic Control in IP Networks) 2002*, Boston, MA, July 29 – August 1 2002.

[27] Z. Duan, Z.-L. Zhang, and Y. T. Hou. Service overlay networks: SLAs, QoS and bandwidth provisioning. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Paris, France, November 2002. Winner of Best Paper Award.

[28] D. Ferrari, A. Banerjea, and H. Zhang. Network support for multimedia: A discussion of the tenet approach. *Computer Networks and ISDN Systems*, 10:1267–1280, July 1994.

[29] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8:368–379, April 1990.

[30] N. Figueira and J. Pasquale. An upper bound on delay for the virtual clock service discipline. *IEEE/ACM Transactions on Networking*, 3(4):399–408, August 1995.

[31] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[32] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[33] L. Georgiadis, R. Guérin, and A. Parekh. Optimal multiplexing on a single link: Delay and buffer requirements. In *Proc. IEEE INFOCOM*, pages 524–532, 1994.

[34] L. Georgiadis, R. Guérin, V. Peris, and K. N. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. *IEEE/ACM Transactions on Networking*, 4(4):482–501, 1996.

[35] A. Girard. *Routing and Dimentioning in Circuit-Switched Networks*. Addison-Wesley, 1990.

[36] S. Jamaloddin Golestani. Congestion-free transmission of real-time traffic in packet networks. In *Proc. IEEE INFOCOM*, pages 527–536, San Francisco, CA, June 1990.

[37] S. Jamaloddin Golestani. A stop-and-go queueing framework for congestion management. In *Sigcomm '90: Communication Architectures and Protocols*, pages 8–18, Philadelphia, PA, September 1990.

[38] L. Golubchik and J. Liu. A fast and accurate iterative solution of a multi-class threshold-based queueing system with hysteresis. In *Proc. ACM SIGMETRICS*, Santa Clara, CA, June 2000.

[39] R. Guérin, H. Ahmadi, and M. Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *IEEE Journal on Selected Areas in Communications*, 9(7):968–981, September 1991.

[40] R. Guérin, S. Blake, and S. Herzog. Aggregating RSVP-based QoS requests. Internet Draft, 1997. Work in Progress.

[41] R. Guérin and L. Gün. A unified approach to bandwidth allocation and access control in fast packet-switched networks. In *Proc. IEEE INFOCOM*, volume 1, pages 1–12 (1A.1), Florence, Italy, May 1992.

[42] J. Hyman, A. Lazar, and G. Pacifici. MARS: The Magnet II real-time scheduling algorithm. In *Sigcomm '91 Conference: Communications Architectures and Protocols*, pages 285–293, Zürich, Switzerland, September 1991.

[43] J. Hyman, A. Lazar, and G. Pacifici. Real-time scheduling with quality of service constraints. *IEEE Journal on Selected Areas in Communications*, 9(7):1052–1063, September 1991.

[44] V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding PHB. RFC 2598, June 1999.

[45] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A measurement-based call admission control for integrated services packet networks. In *Proc. ACM SIGCOMM*, pages 2–13, Cambridge, MA, August 1995.

[46] S. Jamin, S. Shenker, L. Zhang, and D. Clark. An Admission Control Algorithm for Predictive Real-Time Service. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 308–315. IEEE Computer and Communication Societies, November 1992.

[47] C. Kalmanek, H. Kanakia, and S. Keshav. Rate controlled servers for very high-speed networks. In *Proc. IEEE GLOBECOM*, pages 12–20 (300.3), San Diego, CA, December 1990.

[48] F. P. Kelly. Routing in circuit-switched networks: Optimization, shadow prices and decentralization. *Advances in Applied Probability*, 20:112–144, 1988.

[49] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley, 1997.

[50] J. Kurose. Open issues and challenges in providing quality-of-service guarantees in high-speed networks. *Computer Communication Review*, 23(1), January 1993.

[51] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *SIGCOMM*, pages 175–187, 2000.

[52] C. Labovitz, G. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6(5):515–528, 1998.

[53] W. E. Leland, M. S. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1), 1994.

[54] J. Liebeherr and D. E. Wrege. A versatile packet multiplexer for quality-of-service networks. In *Proc. 4th International Symposium on High Performance Distributed Computing (HPDC-4)*, pages 148–155, August 1995.

[55] J. Liebeherr, D. E. Wrege, and D. Ferrari. Exact admission control for networks with a bounded delay service. *IEEE/ACM Transactions on Networking*, 4(6):885–901, 1996.

[56] Z. Mao, R. Govindan, G. Varghese, and R. Katz. Route flap damping exacerbates Internet routing convergence. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, August 2002.

[57] R. Nagarajan and J. Kurose. On defining, computing and guaranteeing quality-of-service in high-speed networks. In *Proc. IEEE INFOCOM*, volume 3, pages 2016–2025 (8C.2), Florence, Italy, May 1992.

[58] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the internet. RFC 2638, July 1999.

[59] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks — the single node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[60] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks — the multiple node case. *IEEE/ACM Transactions on Networking*, 2(2):137–150, 1994.

[61] M. Parulekar and A. M. Markowski. M/G/∞ input processes: A versatile class of models for network traffic. In *Proc. IEEE INFOCOM*, pages 419–426, Kobe, Japan, April 1997.

[62] V. Paxson. End-to-end routing behavior in the Internet. In *Proc. ACM SIGCOMM*, Stanford, CA, August 1996.

[63] V. Paxson and S. Floyd. Wide area traffic: The failure of poisson modeling. In *Proc. ACM SIGCOMM*, pages 257–268, August 1994.

[64] D. Pei, X. Zhao, L. Wang, D. Massey, A. Mankin, S. Wu, and L. Zhang. Improving bgp convergence through consistency assertions. In *INFOCOM 2002*, New York, NY, Jun 2002.

[65] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). RFC 1771, March 1995.

[66] E. C. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. Internet Draft, August 1999. Work in Progress.

[67] S. Savage, T. Anderson, and et al. Detour: a case for informed internet routing and transport. *IEEE Micro*, 19(1):50–59, January 1999.

[68] S. Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communications*, 13(7), September 1995.

[69] S. Shenker, C. Partridge, and R. Guérin. Specification of guaranteed quality of service. RFC 2212, September 1997.

[70] D. Stiliadis. *Traffic Scheduling in Packet-Switched Networks: Analysis, Design, and Implementation*. PhD thesis, Computer Science and Engineering Department, University of California at Santa Cruz, June 1996.

[71] D. Stiliadis and A. Varma. Efficient fair queueing algorithms for packet-switched networks. *IEEE/ACM Transactions on Networking*, 6(2):175–185, 1998.

[72] D. Stiliadis and A. Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, 1998.

[73] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In *Proc. ACM SIGCOMM*, Boston, MA, September 1999.

[74] I. Stoica, H. Zhang, S. Shenker, R. Yavatkar, D. Stephens, A. Malis, Y. Bernet, Z. Wang, F. Baker, J. Wroclawski, C. Song, and R. Wilder. Per hop behaviors based on dynamic packet states. Internet Draft, February 1999. Work in Progress.

[75] A. Terzis, J. Ogawa, S. Tsui, L. Wang, and L. Zhang. A prototype implementation of the two-tier architecture for differentiated services. In *Proceedings of IEEE RTAS'99*, Vancouver, Canada, 1999.

[76] A. Terzis, L. Wang, J. Ogawa, and L. Zhang. A two-tier resource management model for the internet. In *Global Internet 99*, December 1999.

[77] D. Towsley. Providing quality of service in broadband integrated services digital networks. In *Performance Evaluation of Computer and Communication Systems*, pages 560–586. Springer-Verlag, 1993.

[78] Auckland Data Trace. http://pma.nlanr.net/traces/long/auck2.html.

[79] L. Wang, A. Terzis, and L. Zhang. A new proposal of RSVP refreshes. In *Proceedings of IEEE ICNP*, Toronto, Canada, November 1999.

[80] L. Wang, A. Terzis, and L. Zhang. RSVP refresh overhead reduction by state compression. Internet Draft, June 1999. Work in Progress.

[81] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, October 1995.

[82] H. Zhang and D. Ferrari. Rate-controlled static-priority queueing. In *IEEE INFOCOM'93*, pages 227–236, April 1993.

[83] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. In *Proc. ACM SIGCOMM*, pages 19–29, September 1990.

[84] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, pages 8–18, September 1993.

[85] Z.-L. Zhang, Z. Duan, L. Gao, and Y. T. Hou. Decoupling QoS control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In *Proc. ACM SIGCOMM*, Sweden, August 2000.

[86] Z.-L. Zhang, Z. Duan, and Y. T. Hou. Virtual time reference system: A unifying scheduling framework for scalable support of guaranteed services. *IEEE Journal on Selected Areas in Communication*, Special Issue on Internet QoS, December 2000.

[87] Z.-L. Zhang, Z. Duan, and Y. T. Hou. Fundamental trade-offs in aggregate packet scheduling. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Riverside, CA, November 2001.

[88] Z.-L. Zhang, Z. Duan, and Y. T. Hou. On scalable design of bandwidth brokers. *IEICE Transaction on Communications*, E84-B(8), August 2001.

[89] Z.-L. Zhang, Z. Duan, and Y. T. Hou. On scalable network resource management using bandwidth brokers. In To appear in *8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, Florence, Italy, April 2002.

Part V

Appendices

Appendix A

Proofs Related to Chapter 3

A.1 Proofs Related to Networks of Static Earliest Time First Schedulers

As we discussed in Section 3.4, the SETF(0) discipline is a special version of the SETF(Γ) discipline. Therefore in the following we will focus on the proofs for the general SETF(Γ) schedulers, the lemmas and the theorem for SETF(0) will be given as corollaries.

First we will prove a useful lemma which can be applied to both SETF(0) and SETF(Γ) schedulers.

Lemma 9 *Consider a packet p at an SETF scheduler $\mathcal{S}_i$ along the path.* At time a_i^p, *let p^* be the last packet in the busy period which contains packet p such that when p^* arrives at the scheduler the releasing time (at the network edge) of any packet p' in the queue (including the one in service) satisfies $a_1^{p'} \geq a_1^{p^*}$. Then for any packet $p^\#$ that* starts service *during the time interval $(a_i^{p^*}, a_i^p]$, we have*

$$\omega_0^{p^\#} \leq \omega_0^p.$$

Proof: If packet p is the packet p^*, the lemma holds trivially because the time interval in consideration is empty. In the following we assume that p is not p^*. We will prove the lemma by contradiction. Assume that there is at least one packet $p^\#$ that starts being serviced during the time interval $(a_i^{p^*}, a_i^p]$ such that

$$\omega_0^{p^\#} > \omega_0^p.$$

Let t denote the time that the scheduler starts to service packet $p^{\#}$. It is easy to see that $a_i^{p^*} < t \leq a_i^p$. From the property of the SETF scheduler we know that at time t, for any packet p' in the queue, if there is any, we must have

$$\omega_0^{p'} \geq \omega_0^{p^{\#}}.$$

Consider two cases. If no packet with a releasing time smaller than a_1^p arrives during the time $[t, a_i^p]$, then when packet p arrives at the queue, it is the oldest packet in the queue. Therefore p is the packet p^*, which contradicts our assumption that p is not p^*. On the other hand, assume at least one packet with a releasing time smaller than or equal to a_1^p arrives at the scheduler during the time interval $[t, a_i^p]$. Let p'' denote the first of such packets, then

$$a_1^{p''} \leq a_1^p \leq \omega_0^p < \omega_0^{p^{\#}}.$$

Therefore, when packet p'' arrives at the scheduler, all the packets in the scheduler have a releasing time that is larger than that of p''. Thus p'' satisfies the definition of p^*. This contradicts the condition that packet p'' arrives during the interval $(a_i^{p^*}, a_i^p]$.

This completes the proof. ∎

Proof of Lemma 5: Consider the i scheduler $\mathcal{S}_i$ on the path of packet p. For the case $1 \leq i \leq h^*$, (3.17) follows naturally by noting that we can apply the same argument in Section 3.3 for a network of FIFO schedulers. In the following, we assume that $h^* < i \leq h$.

Denote $\mathcal{M}_i^p$ the set of packets that are serviced between (a_i^p, f_i^p]. From the definition of SETF, we know that the packets in the set should have a time stamp no greater than that of the packet p, and should have not stayed in the network longer than τ^*. Formally, for any packet p' belonging to $\mathcal{M}_i^p$, we have

$$\omega_0^{p'} \leq \omega_0^p \text{ and } a_1^{p'} \geq a_i^p - \tau^*. \tag{A.1}$$

Then, we see that,

$$\begin{aligned} Q(a_i^p) &\leq \sum_{p' \in \mathcal{M}_i^p} L^{p'} + L^{max} \\ &\leq \alpha C_i\{\omega_0^p - (a_i^p - \tau^*)\} + \beta C_i + L^{max} \\ &= \alpha C_i\{\tau^* - (a_i^p - \omega_0^p)\} + \beta C_i + L^{max} \\ &\leq \alpha C_i\{\tau^* - (a_i^p - a_{h^*+1}^p)\} + \beta C_i + L^{max}. \end{aligned} \tag{A.2}$$

The last step is because $a^p_{h^*+1} \geq a^p_1 + \Gamma > \omega^p_0$.

■

Note that when $\Gamma = 0$, we have $h^* = 0$. By setting $h^* = 0$ in Lemma 5, we obtain Lemma 3.

We now prove Theorem 3, which states the main properties of the SETF(Γ) scheduler.

Proof of Theorem 3: The proof includes two parts. We first prove that for the given value of τ^*, (3.4) holds, i.e., no packets will experience a delay larger than τ^* before reaching the last hop. In the second part, we prove that the worst-case edge-to-edge delay D^* is indeed bounded.

Part 1. Let $\tau^* = \frac{\beta h^* + \alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{H^*-h^*-1}\}}{(1-\alpha)^{H^*-h^*-1}-\alpha h^*}$. We claim that for any packet p which traverses a path with h hops, the following relationship holds for any $i \leq h$,

$$a^p_i - a^p_1 \leq \tau^*. \tag{A.3}$$

First consider $i \leq h^* + 1$. From Lemma 2 we know that

$$\begin{aligned} a^p_i - a^p_1 &\leq (i-1)(\alpha\tau^* + \beta) \leq \frac{i\beta + i\Delta\{1-(1-\alpha)^{H^*-h^*-1}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*} \\ &\leq \frac{h^*\beta + h^*\Delta\{1-(1-\alpha)^{H^*-h^*-1}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*}. \end{aligned} \tag{A.4}$$

From (3.21), we know $\alpha^{-1}(1-\alpha)^{H^*-h^*-1} > h^*$. Notice that $H^* \geq h^* + 1$, we have $(1-\alpha)^{H^*-h^*-1} \leq 1$. Therefore, $\alpha^{-1} > h^*$. Hence

$$a^p_i - a^p_1 \leq \frac{h^*\beta + h^*\Delta\{1-(1-\alpha)^{H^*-h^*-1}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*} \leq \tau^*.$$

In the following we prove that (A.3) also holds for the case where $h^* + 1 < i \leq h$ by contradiction. Not loss of generality, assume that for the first time in the system there is a packet p^* which violates the relationship at scheduler $i^* \leq h$ (without loss of generality, we assumed here that the path that packet p^* traverses has h hops), i.e., $a^{p^*}_{i^*} - a^{p^*}_1 > \tau^*$. Therefore (A.3) holds for all packets before the time $a^{p^*}_{i^*}$, in particular, we have

$$a^{p^*}_{i^*-1} - a^{p^*}_1 \leq \tau^*$$

From Lemma 6, we have

$$a_{i^*-1}^{p^*} - a_1^{p^*} \leq h^*(\alpha\tau^* + \beta) + \tau^*\{1-(1-\alpha)^{i^*-h^*-2}\} + \alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{i^*-h^*-2}\}$$

Therefore, we have

$$\begin{aligned} a_{i^*}^{p^*} &\leq (1-\alpha)a_{i^*-1}^{p^*} + \alpha(\tau^* + a_{h^*+1}^{p^*}) + \beta + \Delta, \\ &\leq a_1^{p^*} + (1-\alpha)\{h^*(\alpha\tau^*+\beta) + (\tau^* + \alpha^{-1}(\beta+\Delta))\{1-(1-\alpha)^{i^*-h^*-2}\}\} \\ &\quad +\alpha\{\tau^* + h^*(\alpha\tau^*+\beta)\} + \beta + \Delta \end{aligned}$$

After some algebra, we get,

$$\begin{aligned} a_{i^*}^{p} - a_1^{p} &\leq (1-\alpha)\frac{h^*\beta + \alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{i^*-h^*-2}\} + h^*\Delta\{(1-\alpha)^{i^*-h^*-2} - (1-\alpha)^{H^*-h^*-1}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*} \\ &\quad + \frac{\alpha h^*\beta + \beta + \Delta - \alpha h^*\Delta(1-\alpha)^{H^*-h^*-1}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*} \\ &= \frac{h^*\beta + \alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{H^*-h^*-1}\} + (h^*\Delta - \alpha^{-1}(\beta+\Delta))\{(1-\alpha)^{i^*-h^*-1} - (1-\alpha)^{H^*-h^*-1}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*} \\ &\leq \frac{h^*\beta + \alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{H^*-h^*-1}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*} = \tau^*. \end{aligned}$$

The last step comes from the fact that $h^* < \alpha^{-1}$ as we derived earlier and the fact that $(1-\alpha)^{i^*-h^*} - (1-\alpha)^{H^*-h^*-1} \geq 0$. Thus we arrive at a contradiction.

Part 2. In this part we prove the worst-case edge-to-edge delay is indeed bounded. Consider an arbitrary packet p which traverses a path with h hops. From Lemma 6, we have

$$\begin{aligned} f_h^p - a_1^p &\leq h^*(\alpha\tau^*+\beta) + \tau^*\{1-(1-\alpha)^{h-h^*}\} + \alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{h-h^*}\} \\ &< \frac{\beta h^* + \alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{h-h^*}\} + h^*\Delta\{(1-\alpha)^{h-h^*} - (1-\alpha)^{H^*-h^*}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*} \\ &\leq \frac{\beta h^* + \alpha^{-1}(\beta+\Delta)\{1-(1-\alpha)^{H^*-h^*}\}}{(1-\alpha)^{H^*-h^*-1} - \alpha h^*}. \end{aligned} \tag{A.5}$$

■

Again, by setting $h^* = 0$ in the above theorem, we obtain Theorem 2.

A.2 Proofs Related to A Network of DETF(0,1) schedulers

Consider an arbitrary packet p that traverses a path with h hops. Recall that the packet state of packet p is updated according to (3.25) at each scheduler on the path of the packet. Consider an arbitrary scheduler $\mathcal{S}_i$ on the path, we know that the virtual arrival time is ω_{i-1}^p and real arrival time is a_i^p, moreover, the reality check condition (3.26) holds. Define $\mathcal{F}_i$ to be the set of flows traversing the scheduler.

First we prove that $f_i^p \leq \omega_{i-1}^p + \beta + \Delta$ for $0 < \alpha < 1$.

Lemma 10 *For any network utilization level $0 < \alpha < 1$, the departure time of an arbitrary packet p at a scheduler $\mathcal{S}_i$ is bounded by,*

$$f_i^p \leq \omega_{i-1}^p + \beta + \Delta. \tag{A.6}$$

Proof: Consider the scheduler busy period containing packet p. Without loss of generality, assume that the scheduler busy period starts at time 0. Let τ be the last time before the arrival of packet p such that the scheduler *starts* servicing a packet whose virtual arrival time is larger than that of packet p (i.e., ω_{i-1}^p). If such a packet does not exist, set $\tau = 0$, the beginning of the scheduler busy period. Hence $0 \leq \tau \leq a_i^p \leq \omega_{i-1}^p$. Let $t = \omega_{i-1}^p - \tau$. It is easy to see that $t \geq 0$ because $\omega_{i-1}^p \geq a_i^p \geq \tau$.

For each flow $j \in \mathcal{F}_i$, let S^j denote the set of packets of flow j that are serviced after τ but no later than packet p (i.e., they are serviced during the time interval $(\tau, f_i^p]$). From the definition of τ and S^j, we see that for any packet $p^{j,l} \in S^j$, we have $a_i^{p^{j,l}} \geq \tau$ and $\omega_{i-1}^{p^{j,l}} \leq \omega_{i-1}^p$ hold. Notice the reality check condition, we know that $\omega_{i-1}^{p^{j,l}} \geq a_i^{p^{j,l}} \geq \tau$. Therefore, all the packets in S^j must virtually arrive during the interval $[\tau, t + \tau]$, hence from Lemma 7

$$\sum_{j \in \mathcal{F}_i} \sum_{p^{j,l} \in S^j} L^{p^{j,l}} \leq \beta C_i + \alpha C_i t.$$

Note that packet p is included in the left-hand side of the above inequation. Hence packet p must have departed from the scheduler when all the packets in $\sum_{j \in \mathcal{F}_i} S^j$ have been serviced by the scheduler. Furthermore, the packet which is being serviced at time τ (if it exists) has size of at most L^{max}. Therefore, we have

$$f_i^p \leq \tau + \frac{L^{max}}{C_i} + \frac{\sum_{j \in \mathcal{F}_i} \sum_{p^{j,l}}}{C_i} \leq \tau + \Delta + \beta + \alpha(\omega_{i-1}^p - \tau) \leq (1 - \alpha)\tau + \alpha\omega_{i-1}^p + \beta + \Delta$$

Notice that $\tau \leq a_i^p \leq \omega_{i-1}^p$, we have

$$f_i^p \leq (1-\alpha)\omega_{i-1}^p + \alpha\omega_{i-1}^p + \beta + \Delta \leq \omega_{i-1}^p + \beta + \Delta$$

■

Proof of Theorem 4: By assigning $d^* \geq \beta + \Delta$, we see that the reality check condition holds at each scheduler. Moreover, From the above lemma, we have

$$f_h^p \leq \omega_{h-1}^p + \beta + \Delta = \omega_0^p + (h-1)d^* + \beta + \Delta = a_1^p + hd^*.$$

■

A.3 Proofs Related to A Network of DETF(Γ,1) Schedulers

Following a similar procedure as in Appendix A.2, we can prove the following relation holds, for any packet p traversing a path with h hops,

$$f_h^p \leq \omega_0^p + hd^*,$$

where $d^* = \lceil(\alpha\Gamma + \beta + \Delta)/\Gamma\rceil\Gamma$. Note that $\omega_0^p \leq a_1^p + \Gamma$, then we have

$$f_h^p \leq a_1^p + hd^* + \Gamma.$$

A.4 Proofs Related to A Network of DETF(Γ, h^*) Schedulers

In the following we will prove the Theorem 6. First let us recall some notation. Consider an arbitrary packet p which traverses a path with h hops. Let h^* denote the number of schedulers on each segment of the path (the last segment can have less than h^* hops), and assume that there are κ segments on the path, i.e., $\kappa = \lceil \frac{h}{h^*} \rceil$. Consider an arbitrary segment k on the path. Recall that the virtual arrival time of packet p at the segment k is ω_{k-1}^p.Furthermore, the following reality check condition holds,

$$f_{(k-1)h^*}^p \leq \omega_{k-1}^p,$$

where f_0^p is defined as ω_0^p.

Define τ^* such that the following relation holds for any segment k and any packet p,

$$a^p_{kh^*} - \omega^p_{k-1} \leq \tau^*, \tag{A.7}$$

i.e., the delay for any packet to reach the h^*th hop (i.e., the last hop) within any segment is bounded by τ^* compared with the virtual arrival time of the packet at the segment.

Our target is first to prove that $f^p_{kh^*} \leq \omega^p_{k-1} + \Gamma$ for a given *allowable* network utilization and value of Γ. Not loss of generality, let us assume that $f^p_{kh^*} > \omega^p_{k-1}$. Otherwise, if for any packet p on all the segment of its path, $f^p_{kh^*} > \omega^p_{k-1}$ does not hold, then it is easy to see that $f^p_h - \omega^p_0 \leq \kappa\Gamma$ and notice that $\omega^p_0 \leq a^p_1 + \Gamma$, Theorem 6 holds in this case trivially.

Now let us focus on the case where $f^p_{kh^*} > \omega^p_{k-1}$. Let scheduler $\mathcal{S}_{i^*}$ be the scheduler in a segment k on the path where the following condition holds, $a^p_{i^*} \leq \omega^p_{k-1} \leq f^p_{i^*}$. That is, starting from scheduler $\mathcal{S}_{i^*+1}$ (if it exists), the real arrival time of the packet at the scheduler is no later than the virtual arrival time of the packet at the segment. In the following, we will prove the theorem in two parts. First we will derive a bound on $f^p_{i^*}$; then we develop a recursive relation between $f^p_{i^*}$ and f^p_i for $i^* < i \leq kh^*$.

Lemma 11 *Let scheduler $\mathcal{S}_{i^*}$ be defined as above. Then,*

$$f^p_{i^*} \leq \omega^p_{k-1} + \alpha(\tau^* + \Gamma) + \beta + \Delta. \tag{A.8}$$

Proof: Consider the scheduler busy period containing packet p. Without loss of generality, assume that the scheduler busy period starts at time 0. Let τ be the last time before the arrival of packet p such that the scheduler *starts* servicing a packet whose virtual arrival time is larger than that of packet p (i.e., ω^p_{k-1}). If such a packet does not exist, set $\tau = 0$, the beginning of the scheduler busy period. Hence $0 \leq \tau \leq a^p_{i^*} \leq \omega^p_{k-1}$. Let $t = \omega^p_{k-1} - \tau$. It is easy to see that $t \geq 0$ because $\omega^p_{k-1} \geq a^p_{i^*} \geq \tau$.

For each flow $j \in \mathcal{F}_{i^*}$ ($\mathcal{F}_{i^*}$ is defined as before: the set of flows traversing scheduler $\mathcal{S}_{i^*}$), let S^j denote the set of packets of flow j that are serviced after τ but no later than packet p (i.e., they are serviced during the time interval $(\tau, f^p_{i^*}]$). From the definition of τ and S^j, we see that for any packet $p^{j,l} \in S^j$, we have $a^{p^{j,l}}_{i^*} \geq \tau$ and $\omega^{p^{j,l}}_{k-1} \leq \omega^p_{k-1}$ hold (here for all packets we used the subscript k for simplicity even though the current scheduler may not in the kth segment of that flow). Notice that $a^{p^{j,l}}_{i^*} \leq \omega^{p^{j,l}}_{k-1} + \tau^*$ (A.7), we have $\omega^{p^{j,l}}_{k-1} \geq \tau - \tau^*$.

Therefore, all the packets in S^j must virtually arrive during the interval $[\tau - \tau^*, t + \tau]$. Hence from Lemma 7,

$$\sum_{j \in \mathcal{F}_i} \sum_{p^{j,l} \in S^j} L^{p^{j,l}} \leq \beta C_{i^*} + \alpha C_{i^*}(t + \tau^* + \Gamma).$$

Note that packet p is included in the left-hand side of the above inequation. Hence packet p must have departed from the scheduler when all the packets in $\sum_{j \in \mathcal{F}_{i^*}} S^j$ have been serviced by the scheduler. Furthermore, the packet which is being serviced at time τ (if it exists) has size of at most L^{max}. Therefore, we have

$$\begin{aligned} f^p_{i^*} &\leq \tau + \frac{L^{max}}{C_{i^*}} + \frac{\sum_{j \in \mathcal{F}_{i^*}} \sum_{p^{j,l}}}{C_{i^*}} \\ &\leq \tau + \Delta + \beta + \alpha(\omega^p_{k-1} - \tau + \tau^* + \Gamma) \\ &\leq (1 - \alpha)\tau + \alpha\omega^p_{k-1} + \alpha(\tau^* + \Gamma) + \beta + \Delta \end{aligned} \tag{A.9}$$

Notice that $\tau \leq a^p_{i^*} \leq \omega^p_{k-1}$, we have

$$f^p_{i^*} \leq (1 - \alpha)\omega^p_{k-1} + \alpha\omega^p_{k-1} + \alpha(\tau^* + \Gamma) + \beta + \Delta \leq \omega^p_{k-1} + \alpha(\tau^* + \Gamma) + \beta + \Delta$$

■

Next, we will build a relation between the departure time of packet p from scheduler $\mathcal{S}_i$, f^p_i and the departure time of packet p from scheduler $\mathcal{S}_{i^*}$, $f^p_{i^*}$, for $i^* < i \leq kh^*$. First, we will develop some helpful lemmas.

Lemma 12 *Assume scheduler $\mathcal{S}_i$ is on the kth segment of the path of a packet p, where $i^* < i \leq kh^*$. Let C_i be the capacity of the scheduler. Define $Q(a^p_i)$ to be the amount of traffic serviced by the scheduler during the time interval $[a^p_i, f^p_i]$. Then*

$$Q(a^p_i) \leq \alpha C_i(\tau^* + \Gamma) + \beta C_i + L^{max} \tag{A.10}$$

Proof: At time a^p_i, let p^* be the last packet in the busy period which contains packet p such that when p^* arrives at scheduler $\mathcal{S}_i$ the virtual arrival time of any packet p' in the queue (including the packet in service), satisfies

$$\omega^{p'}_{k-1} \geq \omega^{p^*}_{k-1}$$

Note that we can always find such a packet p^* in the busy period because if all other packets do not satisfy the condition, the packet that starts the current busy period certainly does. Following a same argument as in the proof of Lemma 9, we can show that for any packet $p^{\#}$ that *starts being serviced* during the time interval $(a_i^{p^*}, a_i^p]$, we have

$$\omega_{k-1}^{p^{\#}} \leq \omega_{k-1}^p.$$

Therefore,

$$\begin{aligned} Q(a_i^p) &\leq \sum_{j \in \mathcal{F}_i} \{\rho^j(\omega_{k-1}^p - \omega_{k-1}^{p^*} + \Gamma) + \sigma^j\} - (a_i^p - a_i^{p^*})C_i + L^{max} \\ &\leq \alpha C_i(\omega_{k-1}^p - \omega_{k-1}^{p^*} + \Gamma) + \beta C_i - (a_i^p - a_i^{p^*})C_i + L^{max}. \end{aligned} \tag{A.11}$$

The term L^{max} comes from the fact at most one packet is transmission at time $a_i^{p^*}$ which is possilbly have a virtual arrival time that is larger than ω_{k-1}^p.

Notice that $\omega_{k-1}^p \leq f_{i^*}^p \leq a_i^p$, we show below that (A.10) holds.

$$\begin{aligned} Q(a_i^p) &\leq \alpha C_i(a_i^p - \omega_{k-1}^{p^*} + \Gamma) + \beta C_i - (a_i^p - a_i^{p^*})C_i + L^{max} \\ &\leq (\alpha C_i - C_i)(a_i^p - a_i^{p^*}) + \alpha C_i(a_i^{p^*} - \omega_{k-1}^{p^*} + \Gamma) + \beta C_i + L^{max} \\ &\leq \alpha C_i(\tau^* + \Gamma) + \beta C_i + L^{max}, \end{aligned}$$

where the last step is because $\alpha \leq 1$ and $a_i^{p^*} - \omega_{k-1}^{p^*} \leq \tau^*$. ∎

Lemma 13 *Consider a packet p traversing the kth segment of the path. For simplicity we assume that there are h^* hops on the segment. Then, for $i = i^* + 1, i^* + 2, \ldots, h^*$, we have*

$$f_i^p - \omega_{k-1}^p \leq i\{\alpha(\tau^* + \Gamma) + \beta + \Delta\}. \tag{A.12}$$

Proof: From Lemma 12, it is easy to see that, for $m = i^* + 1, i^* + 2, \ldots, i$, we have

$$f_m = a_m + \frac{Q(a_m)}{C_m} \leq a_m + \alpha(\tau^* + \Gamma) + \beta + \Delta.$$

Recursively applying the above equation on $m = i, i-1, \ldots, i^*+1$ we have

$$f_i^p \leq f_{i^*}^p + (i - i^*)\{\alpha(\tau^* + \Gamma) + \beta + \Delta\}.$$

Notice the results in Lemma 11, we get for $i^* + 1 \leq i \leq h^*$

$$f_i^p - \omega_{k-1}^p \leq (i - i^*)\{\alpha(\tau^* + \Gamma) + \beta + \Delta\} + \alpha(\tau^* + \Gamma) + \beta + \Delta \leq i\{\alpha(\tau^* + \Gamma) + \beta + \Delta\},$$

the step comes from the fact that $i^* \geq 1$. ∎

The following theorem states the delay bound within a segment of a path compared with the virtual arrival time to the segment of the path.

Theorem 17 *Let h^* be the length of a segment of a path. If the utilation factor α satisfies the condition $\alpha < \frac{1}{h^*-1}$, then we have that $\tau^* \leq \frac{(h^*-1)(\alpha\Gamma+\beta+\Delta)}{1-(h^*-1)\alpha}$. Moreover, compared with the virtual arrival time of a packet to the segment, the maximum delay within the segment D^* is bounded above by*

$$D^* \leq \frac{h^*(\alpha\Gamma + \beta + \Delta)}{1 - (h^* - 1)\alpha}$$

Proof: The proof includes two parts. We first prove that for the given value of τ^*, (A.7) holds, i.e., no packets will experience a delay larger than τ^* before reaching the last hop compared to the virtual arrival time. In the second part, we prove that the delay of any packet is indeed bounded compared to the virtual arrival time.

Part 1. Let $\tau^* = \frac{(h^*-1)(\alpha\Gamma+\beta+\Delta)}{1-(h^*-1)\alpha}$. We claim that for any packet p which traverses a segment k with h^* hops, the following relationship holds for any $i \leq kh^*$ (note that the kh^*th hop is the last hop on the kth segment),

$$a_i^p - \omega_{k-1}^p \leq \tau^*. \tag{A.13}$$

Otherwise, assume that for the first time in the system there is a packet p^* which violates the relationship at scheduler $i^\# \leq kh^*$ (without loss of generality, we assumed here that the segment that packet p^* traverses is the kth segment), i.e., $a_{i^\#}^{p^*} - \omega_{k-1}^{p^*} > \tau^*$. Therefore (A.13) holds for all packets before the time $a_{i^\#}^{p^*}$, in particular, we have

$$a_{i^\#-1}^{p^*} - \omega_{k-1}^{p^*} \leq \tau^*$$

From Lemma 13, we have

$$a^{p^*}_{i^\#-1} - \omega^{p^*}_{k-1} \leq (i^\# - 2)\{\alpha(\tau^* + \Gamma) + \beta + \Delta\}.$$

Therefore,

$$\begin{aligned} a^p_{i^\#} = f^{p^*}_{i^\#-1} &\leq a^{p^*}_{i^\#-1} + \alpha(\tau^* + \Gamma) + \beta + \Delta \\ &\leq \omega^{p^*}_{k-1} + (i^\# - 2)\{\alpha(\tau^* + \Gamma) + \beta + \Delta\} + \{\alpha(\tau^* + \Gamma) + \beta + \Delta\} \\ &\leq \omega^{p^*}_{k-1} + (i^\# - 1)\{\alpha(\tau^* + \Gamma) + \beta + \Delta\} \end{aligned} \quad \text{(A.14)}$$

After some algebra, we have

$$a^{p^*}_{i^\#} - \omega^{p^*}_{k-1} \leq \frac{(h^* - 1)(\alpha\Gamma + \beta + \Delta)}{1 - (h^* - 1)\alpha} = \tau^*.$$

Thus we arrive at a contradition.

Part 2. In this part we prove that the maximum delay D^* within any segment compared with the virtual arrival time to the entry of the segment is bounded above by, $D^* \leq \frac{h^*(\alpha\Gamma+\beta+\Delta)}{1-(h^*-1)\alpha}$.

Consider an arbitrary packet p which traverses the segment of the path. From Lemma 13, we have

$$f^p_{kh^*} - \omega^p_{k-1} \leq h^*\{\alpha(\tau^* + \Gamma) + \beta + \Delta\} \leq \frac{h^*(\alpha\Gamma + \beta + \Delta)}{1 - (h^* - 1)\alpha}.$$

■

Proof of Theorem 6: By assigning $\Gamma \geq \frac{h^*(\alpha\Gamma+\beta+\Delta)}{1-(h^*-1)\alpha}$, we see the reality check condition holds at the entry of each segment. Consider an arbitrary packet p and assume the it traverses has h hops, let $\kappa = \lceil \frac{h}{h^*} \rceil$. Then

$$f^p_h \leq \omega^p_{\kappa-1} + \Gamma \leq \omega^p_0 + \kappa\Gamma.$$

Notice that $\omega^p_0 \leq a^p_1 + \Gamma$, we complete the proof. ■

Appendix B

Proofs Related to Chapter 4

B.1 Proof of Theorem 7

In this appendix, we will prove Theorem 7 in Section 4.3. This theorem states an important recursive relation between $\Delta_i^{j,k}$ and $\Delta_i^{j,k-1}$. The proof of the theorem is based on two preliminary lemmas.

Lemma 14 below relates the cumulative queueing delay experienced by $p^{j,k}$ up to server i (i.e., $\Delta_i^{j,k}$), to either its queueing delay up to server $i-1$ (i.e., $\Delta_{i-1}^{j,k}$), or the queueing delay experienced by the previous packet $p^{j,k-1}$ up to server i (i.e., $\Delta_i^{j,k-1}$).

Lemma 14 *For $i \geq 1$ and $k \geq 2$,*

$$\Delta_i^{j,k} = \max\{\Delta_{i-1}^{j,k}, \Delta_i^{j,k-1} + i\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}\}. \tag{B.1}$$

In the above, we have defined $\Delta_0^{j,k} = 0$ for all k's. ∎

Proof: First note that as $f_i^{j,1} = a_i^{j,1} + \frac{L^{j,1}}{r^j} = f_{i-1}^{j,1} + \frac{L^{j,1}}{r^j}$, $\Delta_i^{j,1} = 0$ for all i, which is also intuitively obvious.

For any $k \geq 2$, we show (B.1) holds by induction on i.

Basis ($i = 1$). Note that $\Delta_0^{j,k} = 0$.

$$\Delta_1^{j,k} = f_1^{j,k} - (a_1^{j,k} + \frac{L^{j,k}}{r^j}) \;\; = \;\; \max\{a_1^{j,k}, f_1^{j,k-1}\} + \frac{L^{j,k}}{r^j} - (a_1^{j,k} + \frac{L^{j,k}}{r^j})$$

$$
\begin{aligned}
&= \max\{a_1^{j,k}, a_1^{j,k-1} + \frac{L^{j,k-1}}{r^j} + \Delta_1^{j,k-1}\} + \frac{L^{j,k}}{r^j} - (a_1^{j,k} + \frac{L^{j,k}}{r^j}) \\
&= \max\{0, \Delta_1^{j,k-1} + \frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}\}.
\end{aligned}
$$

Inductive Step. Now suppose (B.1) holds true up to server i for $k \geq 2$. We show that it is also true at server $i+1$ for $k \geq 2$.

$$\Delta_{i+1}^{j,k} = f_{i+1}^{j,k} - [a_1^{j,k} + (i+1)\frac{L^{j,k}}{r^j}] = \max\{a_{i+1}^{j,k}, f_{i+1}^{j,k-1}\} + \frac{L^{j,k}}{r^j} - [a_1^{j,k} + (i+1)\frac{L^{j,k}}{r^j}]. \quad \text{(B.2)}$$

Now we consider two cases according to the relationship between the arrival time of packet $p^{j,k}$ and the finish time of $p^{j,k-1}$ at the server $i+1$.

CASE 1: $a_{i+1}^{j,k} \leq f_{i+1}^{j,k-1}$. Then by definition,

$$f_{i+1}^{j,k-1} = \Delta_{i+1}^{j,k-1} + a_1^{j,k-1} + (i+1)\frac{L^{j,k-1}}{r^j}.$$

Therefore, from (B.2), we have

$$\Delta_{i+1}^{j,k} = \Delta_{i+1}^{j,k-1} + (i+1)\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}.$$

Note that, in this case, we have

$$
\begin{aligned}
& \Delta_{i+1}^{j,k-1} + (i+1)\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j} \\
= \;& f_{i+1}^{j,k-1} - [a_1^{j,k-1} + (i+1)\frac{L^{j,k-1}}{r^j}] + (i+1)\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j} \\
= \;& f_{i+1}^{j,k-1} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j}) \\
\geq \;& a_{i+1}^{j,k} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j}) \\
= \;& f_i^{j,k} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j}) \\
= \;& \Delta_i^{j,k}
\end{aligned}
$$

Hence (B.1) holds.

CASE 2: $a_{i+1}^{j,k} > f_{i+1}^{j,k-1}$. Then since

$$a_{i+1}^{j,k} = f_i^{j,k} = \Delta_i^{j,k} + a_1^{j,k} + i\frac{L^{j,k}}{r^j},$$

we have

$$\Delta_{i+1}^{j,k} = \Delta_i^{j,k}.$$

Furthermore,

$$\begin{aligned}
& \Delta_{i+1}^{j,k-1} + (i+1)\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j} \\
= \;& f_{i+1}^{j,k-1} - [a_1^{j,k-1} + (i+1)\frac{L^{j,k-1}}{r^j}] + (i+1)\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j} \\
= \;& f_{i+1}^{j,k-1} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j}) \\
< \;& a_{i+1}^{j,k} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j}) \\
= \;& f_i^{j,k} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j}) \\
= \;& \Delta_i^{j,k}
\end{aligned}$$

Hence (B.1) also holds in this case. This completes the proof of Lemma 14. ∎

From the above proof, we observe that for $i \geq 2$ and $k \geq 2$, if $a_i^{j,k} > f_i^{j,k-1}$, we have

$$\Delta_i^{j,k} = \Delta_{i-1}^{j,k}, \tag{B.3}$$

otherwise,

$$\Delta_i^{j,k} = \Delta_i^{j,k-1} + i\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}. \tag{B.4}$$

Note also that $\Delta_{i+1}^{j,k} - \Delta_i^{j,k}$ represents the queueing delay experienced by packet $p^{j,k}$ at server $i+1$. The following lemma shows that as a packet goes through the servers in the ideal reference system, the queueing delay it experiences at each individual server cannot decrease.

Lemma 15 *For $i \geq 1$ and $k \geq 2$,*

$$\Delta_{i+1}^{j,k} - \Delta_i^{j,k} \geq \Delta_i^{j,k} - \Delta_{i-1}^{j,k}. \tag{B.5}$$

(Note that $\Delta_0^{j,k} = 0$.) ∎

Proof: Proof by induction on k.

Basis($k = 1$). Since $\Delta_i^{j,1} = 0$, the inequality (B.5) holds trivially.

Inductive Step. Now suppose (B.5) holds for $k-1$, $k \geq 2$. We show it also holds for k.

$$\begin{aligned}
\Delta_{i+1}^{j,k} - \Delta_i^{j,k} &= f_{i+1}^{j,k} - [a_1^{j,k} + (i+1)\frac{L^{j,k}}{r^j}] - [f_i^{j,k} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j})] \\
&= f_{i+1}^{j,k} - f_i^{j,k} - \frac{L^{j,k}}{r^j} \\
&= (\max\{a_{i+1}^{j,k}, f_{i+1}^{j,k-1}\} + \frac{L^{j,k}}{r^j}) - (\max\{a_i^{j,k}, f_i^{j,k-1}\} + \frac{L^{j,k}}{r^j}) - \frac{L^{j,k}}{r^j} \\
&= \max\{a_{i+1}^{j,k}, f_{i+1}^{j,k-1}\} - \max\{a_i^{j,k}, f_i^{j,k-1}\} - \frac{L^{j,k}}{r^j}.
\end{aligned}$$

We consider two cases.

CASE 1: $a_i^{j,k} > f_i^{j,k-1}$. Under this condition, we have

$$\begin{aligned}
\Delta_{i+1}^{j,k} - \Delta_i^{j,k} &= \max\{a_{i+1}^{j,k}, f_{i+1}^{j,k-1}\} - a_i^{j,k} - \frac{L^{j,k}}{r^j} \\
&\geq a_{i+1}^{j,k} - a_i^{j,k} - \frac{L^{j,k}}{r^j}.
\end{aligned}$$

Note that for $i > 1$,

$$a_i^{j,k} = f_{i-1}^{j,k} = \Delta_{i-1}^{j,k} + a_1^{j,k} + (i-1)\frac{L^{j,k}}{r^j} \tag{B.6}$$

where as $\Delta_0^{j,k} = 0$, the last equality above also holds for $i = 1$.

On the other hand,

$$a_{i+1}^{j,k} = f_i^{j,k} = \Delta_i^{j,k} + a_1^{j,k} + i\frac{L^{j,k}}{r^j} \tag{B.7}$$

From (B.6) and (B.7), we have

$$\Delta_{i+1}^{j,k} - \Delta_i^{j,k} \geq \Delta_i^{j,k} - \Delta_{i-1}^{j,k}.$$

CASE 2: $a_i^{j,k} \leq f_i^{j,k-1}$. Under this condition, we have

$$\begin{aligned} \Delta_{i+1}^{j,k} - \Delta_i^{j,k} &= \max\{a_{i+1}^{j,k}, f_{i+1}^{j,k-1}\} - f_i^{j,k-1} - \frac{L^{j,k}}{r^j} \\ &\geq f_{i+1}^{j,k-1} - f_i^{j,k-1} - \frac{L^{j,k}}{r^j}. \end{aligned}$$

Since $f_{i+1}^{j,k-1} = \Delta_{i+1}^{j,k-1} + a_1^{j,k-1} + (i+1)\frac{L^{j,k-1}}{r^j}$ and $f_i^{j,k-1} = \Delta_i^{j,k-1} + a_1^{j,k-1} + i\frac{L^{j,k-1}}{r^j}$, we have

$$\begin{aligned} \Delta_{i+1}^{j,k} - \Delta_i^{j,k} &\geq \Delta_{i+1}^{j,k-1} - \Delta_i^{j,k-1} + \frac{L^{j,k-1} - L^{j,k}}{r^j} \\ &\geq \Delta_i^{j,k-1} - \Delta_{i-1}^{j,k-1} + \frac{L^{j,k-1} - L^{j,k}}{r^j} \text{ (from inductive hypothesis)} \\ &= \Delta_i^{j,k} - (i\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}) - \Delta_{i-1}^{j,k-1} + \frac{L^{j,k-1} - L^{j,k}}{r^j} \text{ (from (B.4))} \\ &= \Delta_i^{j,k} - (\Delta_{i-1}^{j,k-1} + (i-1)\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}) \\ &\geq \Delta_i^{j,k} - \Delta_{i-1}^{j,k} \text{ (from Lemma 14)} \end{aligned}$$

■

A direct consequence of Lemma 15 is that if for some i, $1 \leq i \leq h$, $\Delta_{i+1}^{j,k} = \Delta_i^{j,k}$, then $\Delta_{i+1}^{j,k} = \Delta_i^{j,k} = \Delta_{i-1}^{j,k} = \cdots = \Delta_1^{j,k} = \Delta_0^{j,k} = 0$. In other words, for any packet $p^{j,k}$, either $\Delta_i^{j,k} = 0$ for $i = 1, 2, \ldots, h$ (i.e., no queueing delay is experienced by the packet); or there exists i^*, $1 \leq i^* \leq h$, such that $\Delta_i^{j,k} = 0$ for $i = 1, \ldots, i^* - 1$ and $\Delta_i^{j,k} > 0$ for $i = i^*, i^* + 1, \ldots, h$ (i.e., the packet starts experiencing queueing delay at server i^* and onwards). Intuitively, the latter case happens because the packets preceding packet $p^{j,k}$ have eventually accumulated enough delay at server i^* to affect packet $p^{j,k}$ (see Figure 4.4(b)). Applying the above fact to Lemma 14, we see that Theorem 7 holds.

Another consequence of Lemma 15 is the following corollary.

Corollary 18 *For any $p \leq h$ and $k = 1, 2, \ldots$, we have*

$$\frac{\Delta_p^{j,k}}{p} \leq \frac{\Delta_h^{j,k}}{h}. \tag{B.8}$$

Proof: Note that for $k = 1$, since $\Delta_p^{j,1} = \Delta_h^{j,1} = 0$, (B.8) holds trivially. For $k \geq 2$, let $\delta_p^{j,k} = \Delta_p^{j,k}/p$. Then $\Delta_p^{j,k} = p\delta_p^{j,k}$. From Lemma 15, it is easy to see that $\Delta_p^{j,k} - \Delta_{p-1}^{j,k} \geq \delta_p^{j,k}$. Furthermore, for any $i = p+1, \ldots, h$, we have $\Delta_i^k - \Delta_{i-1}^k \geq \Delta_p^{j,k} - \Delta_{p-1}^{j,k} \geq \delta_p^{j,k}$. Hence

$$\Delta_h^{j,k} = \sum_{i=p+1}^{h} (\Delta_i^k - \Delta_{i-1}^k) + \Delta_p^{j,k} \geq (h-p)\delta_p^{j,k} + p\delta_p^{j,k} = h\delta_p^{j,k}.$$

Therefore (B.8) also holds for any $k \geq 2$. ∎

We now apply Theorem 7 to derive the following important corollary.

Corollary 19 *For any $k \geq 1$, and $i = 1, 2, \ldots, h$,*

$$\Delta_i^{j,k} + i\frac{L^{j,k}}{r^j} \leq i\frac{L^{j,max}}{r^j}. \tag{B.9}$$

where $L^{j,max}$ is the maximum packet size of flow j.

Proof: Since $\Delta_i^{j,1} = 0$, the inequality (B.9) holds trivially. Suppose (B.9) holds for $1, 2, ..., k-1$, we show that it also holds for k. Note that if $\Delta_i^{j,k} = 0$, then (B.9) holds trivially. Now consider the case $\Delta_i^{j,k} > 0$. From Theorem 7, we have

$$\begin{aligned} \Delta_i^{j,k} + i\frac{L^{j,k}}{r^j} &= \Delta_i^{j,k-1} + i\frac{L^{j,k-1} - L^k}{r^j} + a_1^{j,k-1} - a_1^k + \frac{L^{j,k}}{r^j} + i\frac{L^{j,k}}{r^j} \\ &= \Delta_i^{j,k-1} + i\frac{L^{j,k-1}}{r^j} + a_1^{j,k-1} - a_1^k + \frac{L^{j,k}}{r^j}. \end{aligned} \tag{B.10}$$

From (4.3) and the inductive hypothesis, we see that (B.9) also holds for k.

∎

B.2 Virtual Shaping Lemma and Its Applications

An important consequence of the virtual spacing property of packet virtual time stamps is the following *Virtual Shaping Lemma*. Intuitively, it states that according to the virtual time, the amount of flow j traffic arriving at server j is "constrained" by its reserved rate r^j. This lemma is critical in designing delay-based core stateless scheduling algorithms that

can provide delay guarantees *without explicit rate control or reshaping within the network core*.

Lemma 16 (Virtual Shaping Lemma) *Consider an arbitrary time interval $[\tau, t]$. Let $\tilde{T}^j$ denote the set of the packets of flow j which virtually arrives during $[\tau, t]$, i.e., $k \in \tilde{T}^j$ if and only if $\tau \leq \tilde{\omega}^{j,k} \leq t$. Let $\tilde{A}^j(\tau, t) = \sum_{k \in \tilde{T}^j} L^k$ be the corresponding amount of traffic of flow j arriving virtually between the time interval $[\tau, t]$. Then*

$$\tilde{A}^j(\tau, t) = \sum_{k \in \tilde{T}^j} L^{j,k} \leq r^j(t - \tau) + L^{j,max}$$

Proof: Let k_0 be the smallest index k of the packets $p^{j,k}$ in $\tilde{A}^j$ and k_1 be the largest index k of the packets $p^{j,k}$ in $\tilde{A}^j$. From $k = k_1$ down to k_0, recursively applying the fact that $\tilde{\omega}^{j,k} \geq \tilde{\omega}^{j,k-1} + \frac{L^{j,k}}{r^j}$, we have

$$t \geq \tilde{\omega}^{j,k_1} \geq \tilde{\omega}^{j,k_1-1} + \frac{L^{j,k_1}}{r^j} \geq \tilde{\omega}^{j,k_0} + \frac{\sum_{k=k_0+1}^{k_1} L^{j,k}}{r^j} \geq \tau + \frac{\sum_{k=k_0+1}^{k_1} L^{j,k}}{r^j}.$$

Hence,

$$\tilde{A}^j(\tau, t) = \sum_{k \in \tilde{T}^j} L^{j,k} = \sum_{k=k_0+1}^{k_1} L^{j,k} + L^{j,k_0} \leq r^j(t - \tau) + L^{j,k_0} \leq r^j(t - \tau) + L^{j,max}.$$

■

Using the Virtual Shaping Lemma, we now prove that the VT-EDF scheduling algorithm we designed in Section 4.5 has the minimum error term $\Psi_{VT\text{-}EDF} = L^{*,max}/C$ (see Theorem 11).

Proof of Theorem 11: Fix an arbitrary packet $p^{j,k}$ from a flow j. We show that (4.34) holds.

Consider the system busy period containing packet $p^{j,k}$. Without loss of generality, assume that the beginning of the system busy period starts at time 0. Let τ be the last time before the arrival of packet $p^{j,k}$ such that the scheduler *starts* servicing a packet whose virtual finish time is larger than that of $p^{j,k}$ (i.e., $\tilde{\nu}^{j,k}$). If such a packet does not exist, set $\tau = 0$, the beginning of the busy period. Hence $0 \leq \tau \leq \hat{a}^{j,k} \leq \tilde{\omega}^{j,k}$.

Let $t = \tilde{\omega}^{j,k} + d^j - \tau$. Since $\tilde{\omega}^{j,k} \geq \hat{a}^{j,k} \geq \tau$ and $d^j \geq d^1 \geq 0$, we have that

$$t = \tilde{\omega}^{j,k} + d^j - \tau \geq d_1 \geq 0. \tag{B.11}$$

For each flow m, $m = 1, 2 \ldots, N$, let S^m denote the set of packets of flow m that are serviced after τ and but no later than packet $p^{j,k}$ (i.e., they are serviced during the time interval $(\tau, \hat{f}^{j,k}]$). From the definition of τ and S^m, we see that for any $p^{m,l} \in S^m$, $\hat{a}^{m,l} \geq \tau$ and $\tilde{\nu}^{m,l} \leq \tilde{\nu}^{j,k}$ hold. Since $\tilde{\omega}^{m,l} \geq \hat{a}^{m,l}$, $\tilde{\nu}^{m,l} = \tilde{\omega}^{m,l} + d^m$ and $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k} + d^j = t + \tau$, we have

$$\tau \leq \tilde{\omega}^{m,l} \leq t + \tau - d^m. \tag{B.12}$$

From (B.12), it is clear that if $d^m > t = \tilde{\omega}^{j,k} + d^j - \tau$, then $S^m = \emptyset$. Therefore for any m such that $S^m \neq \emptyset$, we must have $d^m \leq t$. Applying the Virtual Shaping Lemma to flow m over the time interval $[\tau, t - d^m + \tau]$, we have

$$\sum_{l \in S^m} L^{m,l} \leq L^{m,max} + r^m(t - d^m).$$

Summing over all m and using the schedulability condition (4.33), we have

$$\sum_{m=1}^{N} (\sum_{l \in S^m} L^{m,l}) \mathbf{1}_{\{S^m \neq \emptyset\}} \leq \sum_{m=1}^{N} [L^{m,max} + r^m(t - d^m)] \mathbf{1}_{\{t \geq d^m\}} \leq Ct$$

Note that packet $p^{j,k} \in S^j$. Hence packet $p^{j,k}$ must have departed the scheduler when all the packets in $\cup_{m=1}^{N} S^m$ have been serviced by the server. Furthermore, the packet which is being serviced at time τ (if it exists) has a size of at most $L^{*,max}$. Therefore, we have

$$\begin{aligned} \hat{f}^{j,k} &\leq \tau + \frac{L^{*,max}}{C} + \frac{\sum_{m=1}^{N} (\sum_{l \in S^m} L^{m,l}) \mathbf{1}_{\{S^m \neq \emptyset\}}}{C} \\ &\leq \tau + \frac{L^{*,max}}{C} + t = \tilde{\omega}^{j,k} + d^j + \frac{L^{*,max}}{C} = \tilde{\nu}^{j,k} + \frac{L^{*,max}}{C}. \end{aligned} \tag{B.13}$$

■

For a rate-based scheduler $\mathcal{S}$, an alternative form of the Virtual Shaping Lemma can be established. Let $\tilde{\omega}^{j,k}$ be the virtual time stamp associated with $p^{j,k}$ at the entry point of $\mathcal{S}$.

Define the *virtual eligibility time* $\tilde{e}^{j,k}$ of packet $p^{j,k}$ to be $\tilde{e}^{j,k} = \tilde{\omega}^{j,k} + \delta^{j,k}$. Then the virtual finish time of packet $p^{j,k}$ is equal to

$$\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k} + \tilde{d}^{j,k} = \tilde{e}^{j,k} + \frac{L^{j,k}}{r^j}. \tag{B.14}$$

Using a similar proof as in the proof of (4.15) in Theorem 8, we can show that

$$\tilde{e}^{j,k} \geq \tilde{\nu}^{j,k-1}. \tag{B.15}$$

The intuitive meaning of $\tilde{e}^{j,k}$ can be interpreted as follows. Imagine that there were a *virtual rate controller* attached to the rate-based scheduler $\mathcal{S}$. Packet $p^{j,k}$ arriving at the virtual time $\tilde{\omega}^{j,k}$ would be held at the virtual rate controller for $\delta^{j,k}$ amount of time and released to the server $\mathcal{S}$ at the virtual eligible time $\tilde{e}^{j,k} = \tilde{\omega}^{j,k} + \delta^{j,k}$. The packet were then to finish its service at the scheduler $\mathcal{S}$ by the virtual finish time $\tilde{\nu}^{j,k}$. From (B.15), we see that a packet were never released to be serviced at the scheduler $\mathcal{S}$ before its previous packet had finished its service. Using this observation, we establish the following alternative form of the Virtual Shaping Lemma — the *Virtual Rate Control Lemma*.

Lemma 17 (Virtual Rate Control Lemma) *Consider an arbitrary time interval $[\tau, t]$. We say that packet $p^{j,k}$ of flow j is* virtually eligible *for service during $[\tau, t]$ if $\tilde{e}^{j,k} \geq \tau$ and $\tilde{\nu}^{j,k} = \tilde{e}^{j,k} + L^{j,k}/r^j \leq t$. Let $\tilde{S}^j$ denote the set of the packets of flow j which are virtually eligible for service in $[\tau, t]$. Define $\tilde{W}^j(\tau, t) = \sum_{k \in \tilde{S}^j} L^k$. We refer to $\tilde{W}^j(\tau, t)$ as the* virtual eligible work *of flow j over $[\tau, t)$. Then*

$$\tilde{W}^j(\tau, t) = \sum_{k \in \tilde{S}^j} L^{j,k} \leq r^j(t - \tau). \tag{B.16}$$

Proof: Let k_0 be the smallest index k of the packets p^k in $\tilde{S}^j$ and k_1 be the largest index k of the packets $p^{j,k}$ in $\tilde{S}^j$. From $k = k_1$ down to k_0, recursively applying the fact that $\tilde{e}^{j,k} \geq \tilde{\nu}^{j,k-1} = \tilde{e}^{j,k-1} + L^{j,k-1}/r^j$, we have

$$t \geq \tilde{\nu}^{j,k_1} = \tilde{e}^{j,k_1} + \frac{L^{j,k_1}}{r^j} \geq \tilde{e}^{j,k_0} + \frac{\sum_{k=k_0}^{k_1} L^{j,k}}{r^j} \geq \tau + \frac{\sum_{k \in \tilde{S}^j} L^{j,k}}{r^j}.$$

Hence (B.16) follows. ∎

The Virtual Rate Control Lemma states that the amount of virtual eligible work of a flow over any time interval is always limited by its reserved rate. This lemma enables us to design rate-based core stateless scheduling algorithms that can support rate guarantees *without using explicit rate control within the network core*. One such an example is the $C_{\not S}$VC scheduling algorithm we designed in Section 4.5. We now prove that it has the minimum error term $\Psi_{C_{\not S}VC} = L^{*,max}/C$ (see Theorem 9).

Proof of Theorem 9: Fix an arbitrary packet $p^{j,k}$ from any flow j. We show that (4.31) holds.

Consider the system busy period containing packet p_j^k. Without loss of generality, we assume that the beginning of the system busy period starts at time 0. Let τ be the last time before the arrival of packet $p^{j,k}$ such that the scheduler *starts* servicing a packet whose virtual finish time is larger than that of $p^{j,k}$. (In other words, no packet queued at time τ has a virtual finish time smaller than $\tilde{\nu}^{j,k}$.) If such a packet does not exist, set $\tau = 0$, the beginning of the busy period. Hence $0 \leq \tau \leq \hat{a}^{j,k}$.

For each flow m, $m = 1, 2 \ldots, N$, let S^m denote the set of packets of flow m that are serviced by the scheduler after τ but no later than $\hat{f}^{j,k}$, i.e., they are serviced during the time interval $(\tau, \hat{f}^{j,k}]$. From the definition of τ, we have that for any $p^{m,l} \in S^m$, $\hat{a}^{m,l} \geq \tau$ and $\tilde{\nu}^{m,l} \leq \tilde{\nu}^{j,k}$. Now applying the Virtual Rate Control Lemma to flow m over the time interval $[\tau, \tilde{\nu}^{j,k}]$, we have

$$\sum_{l \in S^m} L^{m,l} \leq \tilde{W}^j(\tau, \tilde{\nu}^{j,k}) \leq r^m(\tilde{\nu}^{j,k} - \tau).$$

Summing over all m and using the schedulability condition, we have

$$\sum_{m=1}^{N} \sum_{l \in S^m} L^{m,l} \leq (\sum_{m=1}^{N} r^m)(\tilde{\nu}^{j,k} - \tau) \leq C(\tilde{\nu}^{j,k} - \tau).$$

Note that packet $p^{j,k} \in S^j$. Hence packet $p^{j,k}$ must have departed the scheduler when all the packets in $\cup_{m=1}^{N} S^m$ have been serviced by the server. Furthermore, the packet which is being serviced at time τ (if it exists) has a size of at most $L^{*,max}$. Therefore,

$$\hat{f}^{j,k} \leq \tau + \frac{L^{*,max}}{C} + \frac{\sum_{m=1}^{N} \sum_{l \in S^m} L^{m,l}}{C} \leq \tilde{\nu}^{j,k} + \frac{L^{*,max}}{C}.$$

■

Theorem 10 in Section 4.5 can also be proved using the Virtual Rate Control Lemma.

Proof of Theorem 10: We prove that a slotted $C\!\!\!/$VC scheduler has an error term $\Psi_{slotted-C\!\!\!/VC} = L^{*,max}/C + \iota$. To proceed, consider an arbitrary packet $p^{j,k}$ from a flow j.

Consider the system busy period containing packet $p^{j,k}$. Suppose that $\tau_p \leq \tilde{\nu}^{j,k} < \tau_{p+1}$. Without loss of generality, we assume that the system busy period starts at time 0. Let τ be the last time before the arrival of packet $p^{j,k}$ such that the scheduler starts servicing a packet from a queue $\tau_{p'}$ such as $\tau_{p'} > \tau_p$. If such a packet does not exist, set $\tau = 0$, the beginning of the busy period. Hence $0 \leq \tau \leq \hat{a}^{j,k}$. For each flow m, $m = 1, 2 \ldots, N$, let S^m denote the set of packets of flow m that are serviced after τ and no later than packet $p^{j,k}$ (i.e., they are serviced during the time interval $(\tau, \hat{f}^{j,k}]$). From the definition of τ, we see that for any $p^{m,l} \in S^m$, $\tilde{\omega}^{m,l} \geq \hat{a}^{m,l} \geq \tau$ and $\tilde{\nu}^{m,l} < \tau_{p+1} = \tau_p + \iota$. Applying the Virtual Rate Control Lemma to flow m over the time interval $[\tau, \tau_p + \iota]$, we have

$$\sum_{l \in S^m} L^{m,l} \leq \tilde{W}^m(\tau, \tau_p + \iota) \leq r^m(\tau_p + \iota - \tau).$$

Summing over all m and using the schedulability condition, we have

$$\sum_{m=1}^{N} \sum_{l \in S^m} L^{m,l} \leq (\sum_{m=1}^{N} r^m)(\tau_p + \iota - \tau) \leq C(\tau_p + \iota - \tau).$$

Note that packet $p^{j,k} \in S^j$. Hence packet $p^{j,k}$ must depart the scheduler after all the packets in $\cup_{m=1}^{N} S^m$ have been serviced by the server. Furthermore, the packet which is being serviced at time τ (if it exists) has a size of at most $L^{*,max}$. Therefore,

$$\hat{f}^{j,k} \leq \tau + \frac{L^{*,max}}{C} + \frac{\sum_{m=1}^{N} \sum_{l \in S^m} L^{m,l}}{C} \leq \tau_p + \iota + \frac{L^{*,max}}{C} \leq \tilde{\nu}^{j,k} + \iota + \frac{L^{*,max}}{C}.$$

■

B.3 An Alternative Definition of Latency-Rate Servers

In this appendix, we show that the alternative definition of latency-rate servers is indeed equivalent to the one given in [70, 72].

Proof of Lemma 8: Consider the q^{th} burst period of flow j. Let $p^{j,1}, p^{j,2}, \ldots, p^{j,m}$ denote the first, the second, ..., and the last packet of flow j arriving during this burst period. Let $\hat{a}^{j,1}, \hat{a}^{j,2}, \ldots, \hat{a}^{j,m}$ and $\hat{f}^{j,1}, \hat{f}^{j,2}, \ldots, \hat{f}^{j,m}$ denote the actual arrival and departure times of these packets. By the definition of latency-rate server, we have that for any t, $\hat{a}^{j,1} \leq t \leq \hat{f}^{j,m}$,

$$W^{j,q}(\hat{a}^{j,1}, t) \geq r^j(t - \hat{a}^{j,1} - \Theta^j). \tag{B.17}$$

For $k = 1, 2, \ldots, m$, let $\nu^{j,k}$ be given in (4.37). We show that (B.17) is equivalent to

$$\hat{f}^{j,k} \leq \nu^{j,k} + \Theta^j - \frac{L^{j,k}}{r^j}. \tag{B.18}$$

We first show that (B.17) implies (B.18). For any $k = 1, 2 \ldots, m$, let $t = \hat{f}^{j,k} - \epsilon$, where $0 < \epsilon < \hat{f}^{j,k} - \hat{f}^{j,k-1}$. Since by time t, the scheduler has not finished servicing packet $p^{j,k}$ yet, we have

$$W^{j,q}(\hat{a}^{j,1}, t) = W^{j,q}(\hat{a}^{j,1}, \hat{f}^{j,k-1}) = \sum_{l=1}^{k-1} L^{j,l}.$$

From (B.17), we have

$$\sum_{l=1}^{k-1} L^{j,l} \geq r^j(t - \hat{a}^{j,1} - \Theta^j). \tag{B.19}$$

On the other hand, $\nu^{j,k-1} - \hat{a}^{j,1} = \frac{\sum_{l=1}^{j,k-1} L^{j,l}}{r^j}$. Therefore

$$t \leq \nu^{j,k-1} + \Theta^j \leq \nu^{j,k} - \frac{L^{j,k}}{r^j} + \Theta^j.$$

Letting $\epsilon \to 0$ yields (B.18).

We now show that (B.18) implies (B.17). For any t, $\hat{f}^{j,k-1} \leq t < \hat{f}^{j,k}$, we have

$$\begin{aligned} W^{j,q}(\hat{a}^{j,1}, t) &\geq W^{j,q}(\hat{a}^{j,1}, \hat{f}^{j,k}) - L^{j,k} \\ &= \sum_{l=1}^{k} L^{j,l} - L^{j,k} \end{aligned}$$

$$
\begin{aligned}
&= r^j(\nu^{j,k} - \hat{a}^{j,1}) - L^{j,k} \\
&\geq r^j(\hat{f}^{j,k} + \frac{L^{j,k}}{r^j} - \Theta^j - \hat{a}^{j,1}) - L^{j,k} \\
&= r^j(\hat{f}^{j,k} - \hat{a}^{j,1} - \Theta^j) \geq r^j(t - \hat{a}^{j,1} - \Theta^j).
\end{aligned}
$$

This completes the proof of Lemma 8. ∎

Appendix C

Proofs Related to Chapter 6

C.1 Proofs of Theorem 14 and 15

We first establish Theorem 14.

Proof Theorem 14: Consider an arbitrary time t, where $t \geq t^*$. Denote the queue size at the edge conditioner at t as $Q(t)$. Clearly if at some point during the time interval $(t^*, t]$ the queue becomes empty, then the queueing delay of any packets of the current macroflow is bounded by $d_{edge}^{\alpha'}$. This is because the lingering effect of the old queue at time t^*, $Q(t^*)$, vanishes whenever the queue becomes empty after t^*.

Now suppose that during the time interval $(t^*, t]$ the queue never becomes empty. Then the queue size at the time t satisfies

$$Q(t) \leq Q(t^*) + A^{\alpha'}(t^*, t) - r^{new}(t - t^*), \tag{C.1}$$

where r^{new} is the reserved bandwidth for the new macroflow at time t^*, which includes the contingency bandwidth, Δr^{ν}. Rewrite the above equation as follows,

$$\begin{aligned}
Q(t) &\leq Q(t^*) + A^{\alpha'}(t^*, t) - r^{new}(t - t^*) \\
&= Q(t^*) + A^{\alpha}(t^*, t) + A^{\nu}(t^*, t) - (r^{\alpha}(t - t^*) + r^{\nu}(t - t^*) + \Delta r^{\nu}(t - t^*)) \\
&= Q(t^*) + A^{\alpha}(t^*, t) - r^{\alpha}(t - t^*) + A^{\nu}(t^*, t) - (r^{\nu}(t - t^*) + \Delta r^{\nu}(t - t^*)) \\
&\leq Q(t^*) + A^{\alpha}(t^*, t) - r^{\alpha}(t - t^*) + P^{\nu}(t - t^*) - (r^{\nu}(t - t^*) + \Delta r^{\nu}(t - t^*)) \quad \text{(C.2)} \\
&= Q(t^*) + A^{\alpha}(t^*, t) - r^{\alpha}(t - t^*) + (P^{\nu} - r^{\nu} - \Delta r^{\nu})(t - t^*) \quad \text{(C.3)}
\end{aligned}$$

In (C.2), we used the fact that $A^{\alpha}(t^*, t) \leq L^{\nu,max} + P^{\nu}(t - t^*)$. For ease of exposition, we omitted the term $L^{\nu,max}$ in the subsequent derivation. The effect of this term can be taken into account by accommodating it into the end-to-end formula (i.e., using $D' = D^{\alpha,req} - L^{\nu,max}$ instead of $D^{\alpha,req}$ in the right hand side of (6.24)).

From (C.3), we see that if $P^{\nu} - r^{\nu} - \Delta r^{\nu} \leq 0$, $Q(t)$ is bounded by the maximum queue size in the old system. In other words, the edge queueing delay is bounded by d_{edge}^{old}.

Note that the rate to clear the old queue is at least Δr^{ν}. Hence by time $t^* + \frac{Q(t^*)}{\Delta r^{\nu}}$, the lingering effect of the old queue vanishes. After that time, the edge queueing delay is bounded by that in the new system, namely, $d_{edge}^{\alpha'}$. This completes the proof of Theorem 14. ∎

Now we consider Theorem 15. Note that by having a contigency bandwidth $\Delta r^{\nu} \geq r^{\nu}$, the new system is serviced with a rate at least equal to the service rate in the old system. Therefore, the queueing delay is bounded by the maximum queueing delay in the old system before the lingering effect of the old queue vanishes. As $\Delta r^{\nu} \geq r^{\nu}$, it is easy to see that the lingering effect of the old queue will vanish after $t^* + \frac{Q(t^*)}{\Delta r^{\nu}}$. Theorem 15 then follows easily.

C.2 Proof of Theorem 16

Suppose that at time τ^* the reserved rate of the macroflow α is adjusted at the edge shaper from r to r'. Let p^{k^*} be the last packet that leaves the edge conditioner before the rate change at τ^*, and p^{k^*+1} be the first packet that leaves the edge conditioner after the rate change at τ^*.

To establish the virtual spacing and reality check properties for the macroflow after the rate change at τ^*, we first prove they hold for packet p^{k^*+1}. That is, the following two equations hold for $i = 1, 2, \ldots, h$, where h is the length of path $\mathcal{P}$:

$$\tilde{\omega}_i^{k^*+1} - \tilde{\omega}_i^{k^*} \geq L^{k^*+1}/r', \tag{C.4}$$

and

$$\hat{a}_i^{k^*+1} \leq \tilde{\omega}_i^{k^*+1}. \tag{C.5}$$

From the definitions of $\tilde{\omega}_i^{k^*+1}$, and $\tilde{\omega}_i^{k^*}$, we have

$$\tilde{\omega}_i^{k^*+1} = \hat{a}_1^{k^*+1} + i(\delta^{k^*+1} + \frac{L^{k^*+1}}{r'}) + D_{tot}^{i-1}$$

and

$$\tilde{\omega}_i^{k^*} = \hat{a}_1^{k^*} + i(\delta^{k^*} + \frac{L^{k^*}}{r}) + D_{tot}^{i-1},$$

where $D_{tot}^{i-1} = \sum_{j=1}^{i-1}(\Psi_j + \pi_j)$.

Therefore, to establish (C.4) we only need to show

$$\hat{a}_1^{k^*+1} + i(\delta^{k^*+1} + \frac{L^{k^*+1}}{r'}) \geq \hat{a}_1^{k^*} + i(\delta^{k^*} + \frac{L^{k^*}}{r}) + \frac{L^{k^*+1}}{r'}.$$

Or, equivalently,

$$\delta^{k^*+1} \geq \delta^{k^*} + \frac{L^{k^*}}{r} - \frac{L^{k^*+1}}{r'} + \frac{\hat{a}_1^{k^*} - \hat{a}_1^{k^*+1} + \frac{L^{k^*+1}}{r'}}{i}$$

On the other hand, we have

$$\delta^{k^*+1} = \frac{\Delta^{k^*+1}}{h} \geq \frac{\Delta^{k^*}}{h} + \frac{L^{k^*}}{r} - \frac{L^{k^*+1}}{r'} + \frac{\hat{a}_1^{k^*} - \hat{a}_1^{k^*+1} + \frac{L^{k^*+1}}{r'}}{h}.$$

Since $i \leq h$, we see that (C.4) holds.

Now we show that (C.5) also holds. When $i = 1$, (C.5) follows trivially from the definition. For $i \geq 2$, we have

$$\hat{a}_i^{k^*+1} \quad = \quad \hat{f}_{i-1}^{k^*+1} + \pi_{i-1} \leq \tilde{\nu}_{i-1}^{k^*+1} + \Psi_{i-1} + \pi_{i-1} = \tilde{\omega}_i^{k^*+1}. \tag{C.6}$$

For packet $p^k, k = k^* + 2, k^* + 3, \ldots$, the proof is similar to that of Theorem 2 in [86] and we omit it here.

Now we prove the upper bound (6.23) on the delay a packet experiences inside a network core after time τ^*. Note that if a packet does not catch up with any packet that is injected into the network core before τ^*, then the delay it experiences is bounded by $h\frac{L^{\mathcal{P},max}}{r'} + D_{tot}^{\mathcal{P}}$. Otherwise, it is bounded by $h\frac{L^{\mathcal{P},max}}{r} + D_{tot}^{\mathcal{P}}$. Combining these two cases, we obtain the delay bound (6.23).

Appendix D

Proofs Related to Chapter 8

D.1 $E(W)$ of the static bandwidth provisioning with penalty

From (8.4) and (8.5), it is easy to see that

$$E[W] = \sum_{r\in R} e_r\bar{\rho}_r - \sum_{l\in L}\Phi_l(c_l) - \sum_{r'\in R}\int\cdot\int_{\{\rho_r\}}\pi_{r'}\rho_{r'}B_{r'}(\{\rho_r\})d\{\rho_r\}. \tag{D.1}$$

Moreover,

$$\begin{aligned}\sum_{r'\in R}\int\cdot\int_{\{\rho_r\}}\pi_{r'}\rho_{r'}B_{r'}(\{\rho_r\})d\{\rho_r\} \quad \leq \quad & \sum_{r'\in R}\int\cdot\int_{\{0\}}^{\{\hat{\rho}_r\}}\pi_{r'}\rho_{r'}B_{r'}(\{\rho_r\})d\{\rho_r\} \\ & + \sum_{r'\in R}\sum_{r''\in R, r''\neq r'}\int_{\hat{\rho}_{r''}}^{\infty}(\int\cdot\int_{\{0\}_{R-r''}}^{\{\infty\}})\pi_{r'}\rho_{r'}B_{r'}(\{\rho_r\})d\{\rho_r\} \\ & + \sum_{r'\in R}\int_{\hat{\rho}_{r'}}^{\infty}(\int\cdot\int_{\{0\}_{R-r'}}^{\{\infty\}})\pi_{r'}\rho_{r'}B_{r'}(\{\rho_r\})d\{\rho_r\}. \end{aligned}\tag{D.2}$$

As $B_{r'}(\{\rho_r\}) \leq B_{r'}(\{\hat{\rho}_r\})$ when $\rho_r \leq \hat{\rho}_r, \forall r$,

$$\int\cdot\int_{\{0\}}^{\{\hat{\rho}_r\}}\pi_{r'}\rho_{r'}B_{r'}(\{\rho_r\})d\{\rho_r\} \leq \int\cdot\int_{\{0\}}^{\{\hat{\rho}_r\}}\pi_{r'}\rho_{r'}B_{r'}(\{\hat{\rho}_r\})d\{\rho_r\} \leq \pi_{r'}B_{r'}(\{\hat{\rho}_r\})\bar{\rho}_{r'}. \tag{D.3}$$

Notice $B_{r'}(\{\rho_r\}) \leq 1$ and the definition of δ, we have (note $r'' \neq r'$)

$$\int_{\hat{\rho}_{r''}}^{\infty}(\int\cdot\int_{\{0\}_{R-r''}}^{\{\infty\}})\pi_{r'}\rho_{r'}B_{r'}(\{\rho_r\})d\{\rho_r\} \leq \int_{\hat{\rho}_{r''}}^{\infty}(\int_0^{\infty}\pi_{r'}\rho_{r'}d\rho_{r'})d\rho_{r''} \leq \pi_{r'}\delta\frac{\bar{\rho}_{r'}}{\hat{\rho}_{r''}}. \tag{D.4}$$

Similarly,

$$\int_{\hat{\rho}_{r'}}^{\infty}(\int\cdot\int_{\{0\}_{R-r'}}^{\{\infty\}})\pi_{r'}\rho_{r'}B_{r'}(\{\rho_r\})d\{\rho_r\}\leq\int_{\hat{\rho}_{r'}}^{\infty}\pi_{r'}\rho_{r'}d\rho_{r'}\leq\pi_{r'}\delta. \tag{D.5}$$

Substitute (D.3), (D.4), and (D.5) into (D.2), and then recursively into (D.1), we have

$$E(W)\geq\sum_{r\in R}e_r\bar{\rho}_r-\sum_{l\in L}\Phi(c_l)-\sum_{r\in R}\pi_r\bar{\rho}_rB_r(\{\hat{\rho}_r\})-\sum_{r\in R}\pi_r\delta(1+\sum_{r'\in R,r'\neq r}\frac{\bar{\rho}_r}{\hat{\rho}_{r'}}). \tag{D.6}$$

D.2 Approximate Model of the Dynamic bandwidth Provisioning

The exact solution to the approximate model (8.17) is still difficult to solve. As in the case of the static bandwidth provisioning problem, we will derive a lower bound on $E[\tilde{W}]$, and solve the dynamic bandwidth provisioning problem with respect to the lower bound.

For simplicity of exposition, we consider the special case where the bandwidth cost functions are linear, i.e., for any $l\in L$, $\Phi_l(c_l)=\phi_lc_l$ and $\Phi'_l(\Delta c_l)=\phi'_l\Delta c_l$, where $\phi_l\leq\phi'_l$ for any l. The analysis of this subsection can be extended to the more general bandwidth cost functions.

Observe that

$$\begin{aligned}\int\cdot\int\Phi'_l(\Delta c_l)d\{\rho_r\} &= \phi'_l\int\cdot\int\Delta c_ld\{\rho_r\}=\phi'_l\int\cdot\int_{\{\rho_l\geq\eta_lc_l\}}(\frac{\rho_l}{\eta_l}-c_l)d\{\rho_r\}\\ &= \phi'_l\int\cdot\int_{\{\rho_l\geq\eta_lc_l\}}\frac{\rho_l}{\eta_l}d\{\rho_r\}-\phi'_lc_lPr\{\rho_l\geq\eta_lc_l\}.\end{aligned} \tag{D.7}$$

Since

$$\int\cdot\int_{\{\rho_l\geq\eta_lc_l\}}\frac{\rho_l}{\eta_l}d\{\rho_r\}=\int\cdot\int_{\{\rho_l\geq\eta_lc_l\}}\frac{\sum_{l\in r}\rho_r}{\eta_l}d\{\rho_r\}\leq\int\cdot\int\frac{\sum_{l\in r}\rho_r}{\eta_l}d\{\rho_r\}=\frac{\sum_{l\in r}\bar{\rho}_r}{\eta_l}, \tag{D.8}$$

we have

$$E[\tilde{W}]\geq\sum_r e_r\bar{\rho}_r-\sum_l\phi_lc_l-\sum_l\phi'_l\frac{\sum_{l\in r}\bar{\rho}_r}{\eta_l}+\sum_l\phi'_lc_lPr\{\rho_l\geq\eta_lc_l\}. \tag{D.9}$$

Denote the right-hand side of (D.9) by $\tilde{U}$. Given this lower bound $\tilde{U}$ on $E[\tilde{W}]$, we can use the solution to the following optimization problem as an approximation to the original

optimization problem (8.17):

$$\max_{\{c_l\}} \tilde{U}. \tag{D.10}$$

The necessary condition on the optimal solution $\{c_l^*\}$ to (D.10) is that the gradient of $\tilde{U}$ with respect to $\{c_l\}$ vanishes at $\{c_l^*\}$, i.e., $\nabla \tilde{U} = 0$. Note that for any $l \in L$,

$$\begin{aligned}
\frac{\partial \tilde{U}}{\partial c_l} &= -\phi_l + \phi_l' \frac{\partial}{\partial c_l}\{c_l Pr\{\rho_l \geq \eta_l c_l\}\} \\
&= -\phi_l + \phi_l' \left\{ Pr\{\rho_l \geq \eta_l c_l\} + c_l \frac{\partial}{\partial c_l}\{Pr\{\rho_l \geq \eta_l c_l\}\} \right\} \\
&= -\phi_l + \phi_l' \{Pr\{\rho_l \geq \eta_l c_l\} - \eta_l c_l d\{\rho_l = \eta_l c_l\}\}
\end{aligned} \tag{D.11}$$

where the last inequality follows from the fact that for a cumulative distribution function $F(x)$ with the probability density function $f(x)$, i.e., $F(x) = \int_{-\infty}^{x} f(u)du$, $\frac{dF(x)}{dx} = f(x)$.

Hence we have the following necessary condition on the optimal solution $\{c_l^*\}$

$$\phi_l = \phi_l'(Pr\{\rho_l \geq \eta_l c_l^*\} - \eta_l c_l^* d\{\rho_l = \eta_l c_l^*\}), \tag{D.12}$$

for every $l \in L$. The equations (D.12) may be difficult to solve in practice. In the following we derive an upper bound on c_l^*, which has a nice intuitive interpretation and leads to a simple guideline for choosing the value for the statically reserved bandwidth on each link.

Let c_l' be such that $Pr\{\rho_l > \eta_l c_l'\} = \phi_l/\phi_l'$. Then

$$Pr\{\rho_l \geq \eta_l c_l^*\} \geq Pr\{\rho_l \geq \eta_l c_l^*\} - \eta_l c_l^* d\{\rho_l = \eta_l c_l^*\} = \phi_l/\phi_l' = Pr\{\rho_l > \eta_l c_l'\}. \tag{D.13}$$

As $Pr\{\rho_l > \eta_l c_l\}$ is a decreasing function of c_l, we have

$$c_l^* \leq c_l'. \tag{D.14}$$