Copyright

by

Jasleen Kaur Sahni

2002

The Dissertation Committee for Jasleen Kaur Sahni
certifies that this is the approved version of the following dissertation:

Scalable Network Architecures for Providing Per-flow Service Guarantees

Committee:

Harrick M. Vin, Supervisor

Lorenzo Alvisi

Mike Dahlin

Gustavo DeVeciana

Jorg Liebeherr

Scalable Network Architecures for Providing Per-flow Service Guarantees

by

Jasleen Kaur Sahni, B.Tech., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2002

To Babaji

Acknowledgments

My stay in Austin as a graduate student has been extremely rewarding and fun. I will make an attempt to acknowledge people that have played a role in my life during this stay—my apologies to those I may miss.

It is not possible for me to mention in just a few lines all that I have learned from Harrick Vin, my advisor. His attitude that nothing but the best is acceptable, has worked wonders on the quality of my research. His constant feedback and questioning have not only contributed greatly to this dissertation, but also taught me how to become a better researcher. I will always cherish his mentoring.

I thank Lorenzo Alvisi, Mike Dahlin, Gustavo de Veciana, and Jorg Liebeherr for serving on my dissertation committee. I am grateful for their constructive comments and valuable insights. I would like to especially thank Lorenzo and Mike for additionally helping me out during the interview process, and Gustavo for offering great courses that were instrumental in making better my understanding of networks. Special thanks are due to Jorg for reading my proposal and dissertation at very short notices.

Life in Austin has been enriched by several people I have interacted with, both professionally and personally. I have benefited from many discussions with colleagues during my initial years in the Distributed Multimedia Computing Laboratory (DMCL)—Sergey Gorinsky, Pawan Goyal, Xingang Guo, Amir Husain, Jayaram Mudigonda, Scott Page, Sriram Rao, Prashant Shenoy, Renu Tiwari and T.R.

Viswanathan—and during the last two years in the Laboratory for Advanced Systems Research (LASR)—Puneet Chopra, Ben Hardekopf, Sugat Jain, Ravi Kokku, Taroon Mandhana, Taylor Riche, and Praveen Yalagandula. I cherish these interactions and am grateful for their company. I additionally thank Pawan Goyal and Prashant Shenoy for setting wonderful examples for me to follow. I thank Sergey and Scott for always being there when needed and Scott, additionally, for Lloyd. I thank Deepa and Mythri for all the giggles, joy, and comfort they brought to me. Most of all, I thank Jayaram for his friendship—he has been a never-ending source of indispensable help, encouragement, advice, and inspiration.

I am grateful for having a family that has always given me tremendous love and encouragement during my entire life, despite going through very difficult times. My brother, Anup, continues to be one of my greatest mentors. I am indebted to my dear husband, Darshan, for bearing with a smile on his face, the tough aspect of travel during two years of separation, and for helping me focus on this dissertation through his love, patience, and encouragement. Finally, I am grateful to You, Babaji, for everything.

Jasleen Kaur Sahni

The University of Texas at Austin
August 2002

Scalable Network Architecures for Providing Per-flow Service Guarantees

Publication No. ____________________

Jasleen Kaur Sahni, Ph.D.
The University of Texas at Austin, 2002

Supervisor: Harrick M. Vin

The last decade in providing Internet service was all about building high-bandwidth networks. Two requirements, in contrast, drive the design of next-generation networks: (1) the need for richer service semantics, which stems from the fact that the Internet has seen a rapid emergence of many applications with stringent timeliness constraints that can greatly benefit from end-to-end guarantees on delay, jitter, and throughput; and (2) the need to support large bandwidth networks, which stems from the projected manifold increase in link speeds. Unfortunately, these two requirements are often conflicting. Proposals to provide per-flow service guarantees require the use of complex resource management mechanisms in routers; whereas increase in link speeds mandates the simplification of routers to enable them to operate at high link speeds. The goal of this dissertation is to design a network architecture that meets the above requirements of *scalability* and *providing end-to-end per-flow service guarantees* simultaneously.

Past efforts design network architectures that are either scalable or rich in their service offerings, but not both. Conventional network architectures use the First-in-First-Out (FIFO) link scheduler in routers which, although scalable, fails to provide per-flow service guarantees in the presence of bursty traffic. The *Integrated Services* (IntServ) network architecture, in contrast, enables a network to provide per-flow service guarantees by requiring all routers to employ sophisticated per-flow scheduling algorithms. These scheduling algorithms, however, require routers to perform per-packet flow classification and maintain per-flow scheduling state, which limits their scalability, especially in the core of networks that carry a large number of flows. In this dissertation, we explore the following two design philosophies to achieve simultaneously our objectives of *scalability* and *richness*.

- FIFO networks are scalable, but do not provide guarantees on end-to-end delay and throughput in the presence of bursty traffic. A natural approach, therefore, to providing richer services in scalable FIFO networks is to ask: *can traffic conditioning mechanisms that prevent bursty traffic from entering the network enable FIFO networks to provide per-flow service guarantees?*

- IntServ networks provide per-flow service guarantees, but impose per-flow computational overheads in routers. The natural question of interest is: *is it possible to eliminate complexity from IntServ mechanisms while retaining their strong service semantics?*

In this dissertation, we answer both of the questions raised above. First, we evaluate the efficacy in providing per-flow service guarantees of *constant bit-rate* (CBR) traffic conditioning used in conjunction with FIFO networks. We find that under asymptotic conditions of network utilization and path length, CBR flows may experience significantly high delays in FIFO networks. Our results indicate that CBR shaping is effective in providing performance guarantees to flows, only in environments where the amount of such premium traffic does not exceed a small percentage of the total link capacities.

Second, we develop a network architecture that provides per-flow service guarantees similar to IntServ networks, but without requiring per-flow state or per-packet flow classification in the core routers of the network. We do this in two steps: (1) we understand what end-to-end guarantees are provided by core-stateful networks, and (2) we design core-stateless networks that provide similar guarantees. We instantiate our core-stateless architecture on a programmable network testbed and find that it can be implemented in routers with complexity similar to that of current FIFO networks.

The key contributions of this dissertation include: (1) a comprehensive experimental analysis of the performance of CBR flows in FIFO networks; (2) the *first* tight end-to-end fairness analysis of fair queuing networks; (3) a methodology to transform algorithms from the Guaranteed Rate class of core-stateful algorithms to a core-stateless version that provides the *same* upper bounds on end-to-end delay; (4) the *first* work-conserving core-stateless network that provides deterministic end-to-end throughput guarantees; (5) the *first* work-conserving core-stateless network that provides deterministic end-to-end fairness guarantees; and (6) the first performance analysis of networks that provides per-flow service guarantees, on a programmable router platform.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Opportunity

Next-generation networks will be required to provide to flows performance guarantees with respect to end-to-end delay, throughput, fairness, and jitter. To see why, consider the requirements imposed by emerging network applications.

- *Real-time applications* such as a stock-quote delivery service over the Internet, require that information reaches clients shortly after it gets published at the server; otherwise, it ceases to be of much value to the clients. Similarly, mission-critical applications—for instance, a space-shuttle monitoring system—may have data distributed across various locations. Any updates made to the data at one location will need to be propagated to the other locations in a very short amount of time. In an enterprise setting, where a file system is shared by developers distributed across a network, it may be imperative that any file-system updates be propagated across the enterprise almost instantaneously. To support such applications, networks need to guarantee small end-to-end delays.

- *Multimedia applications* stream audio and video data across the network. To ensure high quality output, the data needs to be displayed at the destination in a timely manner. In order to ensure data availability at all times in spite of variations in end-to-end network delay (jitter), these applications may initially buffer some amount of data. The buffer space requirement, however, can grow very large if the jitter is large. Media streaming applications therefore require that network jitter be bounded and small.

  Video streaming applications, in addition to jitter guarantees, require that networks deliver video frames at the same rate at which they need to be displayed. These high-bandwidth applications, consequently, impose large throughput requirements on the network service, that need to hold even at the short time-scales at which video frames are displayed.

To support applications of the above kind, networks need to provide, on a per-application-flow basis, guarantees on the end-to-end network service.

Providing such per-flow guarantees allows a network service provider to offer richer services than what are offered in the Internet today. Given that basic network connectivity is commoditized, such an offering enables providers to differentiate their service from competitors. Furthermore, when used in conjunction with pricing schemes that charge more for the premium service classes, it provides a means for increasing revenues. Thus, both application needs and economic considerations necessitate that next-generation networks provide end-to-end service guarantees.

1.2 The Challenge

The design of network architectures for providing per-flow service guarantees is faced with the following challenges:

1. *Link capacities are increasing rapidly.*

Capacity	Packet Size (Bytes)	Per-packet Time
10 Mbps Ethernet	64 – 1518	$67.2 - 1240 \mu s$
100 Mbps Ethernet	64 – 1518	$6.72 - 124 \mu s$
1 Gbps Ethernet	64 – 1518	$672 ns - 12.4 \mu s$
155 Mbps (OC3)	53	$3.3 \mu s$
622 Mbps (OC12)	53	$833 ns$
2.45 Gbps (OC48)	53	$208 ns$
9.6 Gbps (OC192)	53	$52 ns$

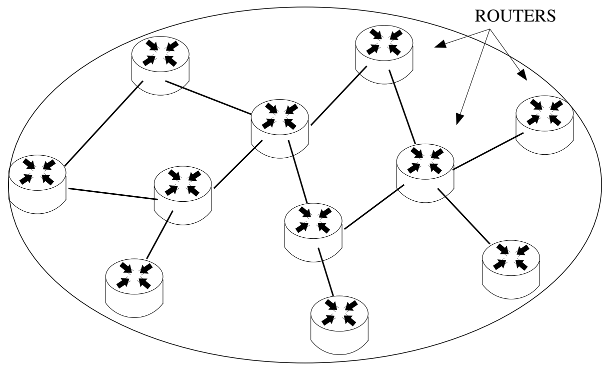
Table 1.1: Per-packet time budget at different link speeds

Providing per-flow guarantees often requires networks to maintain per-flow state and perform per-packet computations. Unfortunately, the amount of time available to process packets in network nodes is decreasing over time. This is because link speeds are increasing by around 100% every year [15], whereas processing speeds are expected to increase at less than 50% [3]. Table 1.1 shows the inter-arrival time of packets for different link speeds. As link capacities grow up to the order of *giga-bits per second*, packets arrive at time-scales of *nano-seconds*. Hence, it is important to minimize the amount of computation performed for each packet, to allow network nodes to scale to high-speed links.

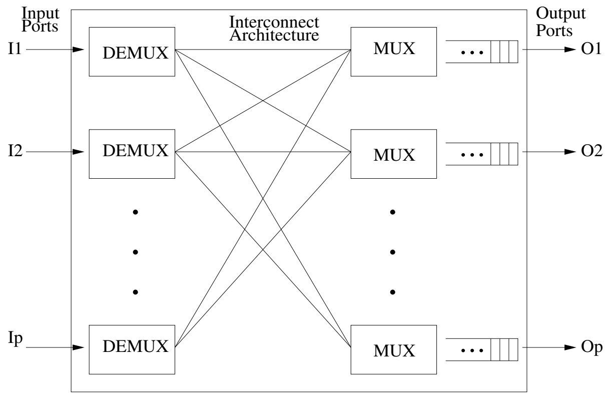
2. *Internet traffic demands are increasing rapidly.*

   Traffic in the Internet is also estimated to increase by about 100% per year [15, 33]. To meet the rising traffic demands, networks must utilize their resources efficiently. Further, routers must scale well as the number of application flows in the network increases.

To summarize, the design of next-generation networks has to be guided by three requirements: (1) providing per-flow guarantees on end-to-end service, (2) scaling to high-speed links and large number of flows, and (3) utilizing resources efficiently. The objective of this dissertation is to design network architectures that meet all of these requirements simultaneously.



(a)



(b)

Figure 1.1: Typical Architectures of (a) a network, and (b) a router

In the rest of this chapter, we first describe the state of the art in existing network architectures and evaluate each with respect to the above set of requirements. We then derive the two research directions we pursue and describe the contributions made in this dissertation.

1.3 State of the Art

1.3.1 Background

Routers[1] are building blocks for networks. Figure 1.1(a) depicts the high-level architecture of a typical wide-area packet-switched network. For each packet that arrives on an input link, the router determines the *next* hop on the end-to-end path of that packet, and transmits the packet on the corresponding output link.

Figure 1.1(b) depicts a router in detail. Data traffic arriving on different input links could be destined to depart on a common outgoing link. Since traffic may arrive *simultaneously* on several input links, the router may have to buffer the packets at the output link[2], and use a packet scheduling algorithm to determine

[1]In this dissertation, we use the terms router, switch, node, and hop, interchangeably.

[2]The above description is for an *output-buffered* switch. Routers may buffer packets at the input

the order in which they are transmitted. The choice of the scheduling algorithm affects the ability of the router to provide service guarantees, such as bounds on the maximum queuing delay suffered, or minimum throughput received by packets from a given flow.

We next consider the state of the art in existing network architectures, based on the choice of packet scheduling algorithms at different routers in the network.

1.3.2 Network Architectures

Conventional First-in-first-out Networks Routers in today's networks use the *First-in-first-out* (FIFO) link scheduling algorithm, that simply transmits packets in the same order that it receives them. FIFO is the simplest known scheduling algorithm and scales well as the number of flows in the network increase. However, it is well-known that FIFO networks do not provide service guarantees to flows. This is because FIFO routers do not protect flows from each other during times of congestion; whenever a burst of packet arrives in one flows, it can cause significant queuing delays to packets of all other flows.

Integrated Services Networks The *Integrated Services* (IntServ) network architecture enables a network to provide per-flow service guarantees [54]. In order to do so, this architecture (1) uses end-to-end signaling to set up packet classification and reservation state on all routers, (2) uses admission control to ensure that the amount of resources reserved does not exceed the available resources, and (3) employs sophisticated per-flow scheduling algorithms at *all* routers. Over the past decade, there has been a considerable amount of research on designing link scheduling algorithms [6, 18, 22, 26, 64, 65] that provide service differentiation and per-flow guarantees on the delay, jitter, throughput, and fairness properties of a net-

link itself, especially if the inter-connect between the input and output links does not operate fast enough to keep up with the sum of input link speeds. Such routers are referred to as *input-buffered*.

work. However, these scheduling algorithms require routers to perform per-packet flow classification, maintain per-flow scheduling state and perform packet sorting to determine the order of transmission of packets. The complexity of these operations increases with the number of flows. They can therefore impose a significant per-packet computational overhead on routers in high-speed networks—especially routers in the core of networks, that carry a large number of flows. Routers in the IntServ architecture, therefore, may not scale well to high-speed links.

Differentiated Services Networks The *Differentiated Services* (DiffServ) network architecture has been proposed recently to provide network services in a scalable manner [44]. This architecture achieves scalability by offering only a fixed number of traffic classes, and letting each flow subscribe to a class. Complex per-flow classification and conditioning functions are implemented only at the edge routers of a network (which process lower volumes of traffic and lesser number of flows). The core routers provide service differentiation inside the network only across traffic classes, and not on a per-flow basis [13, 30, 31, 43]. By limiting the number of traffic classes to a small, fixed value, this architecture achieves scalability of core routers. However, core routers in this architecture do not protect flows within a traffic class from each other, and hence, do not provide per-flow service guarantees similar to IntServ networks.

To summarize, FIFO and DiffServ networks do not provide per-flow service guarantees, whereas IntServ networks do not scale well. Existing network architectures, therefore, only partially meet our design requirements of *service richness* and *scalability*; none of them meets these requirements simultaneously.

1.4 Research Methodology

Observe that the FIFO and IntServ architectures lie at opposite ends of a spectrum—FIFO networks are scalable, but do not provide per-flow service guarantees; IntServ networks provide rich services, but do not scale well to large number of flows and high-speed links. To design networks that meets all of our requirements simultaneously, therefore, the following two research directions can be followed:

- Enable scalable FIFO networks to provide service guarantees.

  Recall that FIFO networks do not provide service guarantees in the presence of bursty traffic. A natural approach to providing richer services in scalable FIFO networks is to ask: *can traffic conditioning mechanisms that prevent bursty traffic from entering the network enable FIFO networks to provide per-flow service guarantees?*

- Simplify router mechanisms in the service-rich IntServ architecture to make it scalable.

  Recall that core routers in the IntServ network do not scale due to the need to maintain per-flow state and perform per-flow packet classification. The natural question of interest is: *is it possible to provide end-to-end service guarantees, similar to those provided by IntServ networks, but without maintaining or using per-flow state in the core routers?*

In this dissertation, we explore both of these research directions. First, we evaluate the efficacy in providing per-flow service guarantees of *constant bit-rate* (CBR) traffic conditioning used in conjunction with FIFO networks. Second, we develop a network architecture that provides per-flow service guarantees similar to IntServ networks, but without requiring per-flow state or per-packet flow classification in the core routers of the network.

In what follows, we briefly summarize our contributions in both of these directions.

1.5 Summary of Contributions

1.5.1 Evaluation of FIFO Networks

Past Work: Past work on the evaluation of the performance of CBR flows in FIFO networks can be divided into three categories: (1) Bounds on end-to-end delay experienced by CBR flows in FIFO networks have been derived in [4, 12, 39]. These bounds, however, are valid only under limiting network and traffic conditions, and are not applicable to general and realistic FIFO networks. (2) Past analytical work on modeling the end-to-end delay performance of CBR flows in FIFO networks [19, 28, 42, 48, 50, 55] is complex and computationally intensive; consequently, it does not yield a closed-form characterization of the performance of CBR flows. (3) Past experimental studies that measure the end-to-end delay and jitter experienced by CBR flows using simulations, are limited to networks that transmit fixed-sized packets.

Our Approach We address the limitations of past analytical and simulation studies on the performance of CBR flows in FIFO networks in two steps: (1) we develop an analytical model that yields a closed-form characterization of the end-to-end performance of CBR flows in FIFO networks under *asymptotic* conditions of network utilization and path length; and (2) we conduct simulations to verify the set of *non-asymptotic* and realistic conditions under which the results of the model continue to hold.

Analytical Results: We model the inter-arrival time between packets at the destination for individual flows (including CBR flows) under *asymptotic* conditions of network utilization and number of hops traversed [60]. Our model yields the following key insights:

1. The variance in inter-arrival times for individual flows tends to infinity in the asymptotic case, indicating that flows become *heavy-tailed* after traversing a network of FIFO routers.

2. The aggregation of flows becomes a long-range dependent self-similar process.

The occurrence of heavy-tails in the inter-arrival times of flows implies that a non-negligible fraction of packets experience significantly large delays. Since the model results are valid even for CBR flows, it follows that shaping flows to CBR at the edge of the network does not result in a low-delay performance in networks that operate at high levels of utilization.

Experimental Results: Our results indicate that:

1. CBR flows can become heavy-tailed after traversing $10 - 20$ hops in networks operating at utilization levels of $40 - 50\%$, or after traversing just $4 - 5$ hops in networks operating at $80 - 90\%$ utilization.

2. The difference in the performance of CBR flows in the FIFO and IntServ architectures is significant in networks with heterogeneous packet sizes (like the current Internet), and those that operate at moderate to high levels of utilization (greater than $50 - 60\%$).

From the above studies, we conclude that FIFO scheduling is inadequate to design scalable networks that devote a significant fraction of available resources to support customers that require per-flow service guarantees.

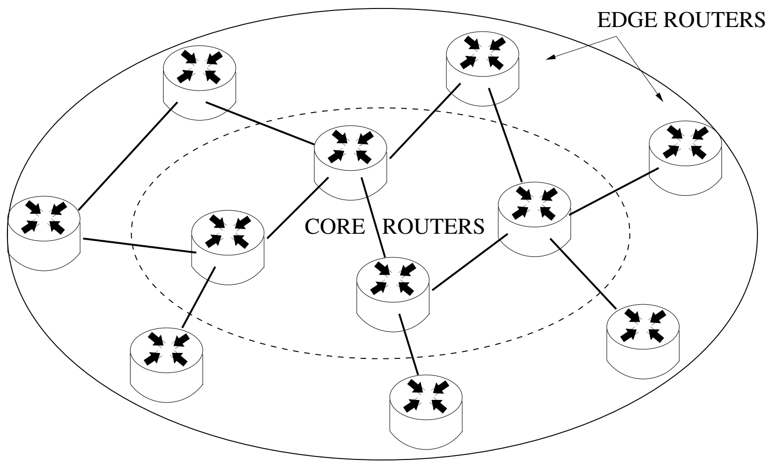


Figure 1.2: Network Model

1.5.2 Design of Core-stateless Guaranteed Services Networks

Problem Formulation

Figure 1.2 depicts edge and core routers in a typical network. The property of not maintaining any per-flow state in the core routers is known as being *core-stateless* [57]. The design of a core-stateless architecture is based on the following simple observations:

- Core routers of a network are connected with high-speed links, and carry a large number of flows. Hence, it is not feasible for these routers to maintain per-flow state and perform packet classification, and yet operate at high link speeds.

- Edge routers of a network operate on low-speed access links, and carry a smaller number of flows. Hence, it is feasible for these routers to maintain and use per-flow state.

A core-stateless architecture therefore promises to scale well as link speeds and traffic increases in the core of networks. Many schemes have been proposed in the recent past to provide service guarantees in a core-stateless architecture. We briefly describe these next.

Past Work Depending on the type of service guarantees provided by core-stateless schemes, these can be classified as those that provide *statistical* guarantees over large time-scales, and those that provide *deterministic* per-flow service guarantees.

In the first category, fall a large number of schemes that provide *approximate* fairness in the long-term throughput achieved by different flows [10, 14, 46, 58]. Most of these schemes employ preferential dropping mechanisms designed to drop packets from flows roughly in proportion to their sending rates. At times of congestion, therefore, more packets are expected to be dropped from flows with larger bit-rates, thereby achieving fairness in the steady-state. However, such fairness is assured only at very large time-scales; no guarantees on fairness or throughput are provided to short-lived flows or during specific intervals of interest in the life-span of long-lived flows.

In contrast, the *Core Jitter Virtual Clock* (CJVC) scheduling algorithm provides deterministic upper bounds on end-to-end network delay [59]. CJVC is the core-stateless version of Jitter Virtual Clock, that provides *exactly* the same end-to-end delay guarantees while maintaining and using per-flow state only at the edge routers of the network. Unfortunately, a CJVC network is non work-conserving, which results in high average delays and limits the extent to which the network benefits from *statistical multiplexing* gains. This is because non-work-conserving algorithms shape the incoming traffic to the maximum of the reserved rate and sending rate for that flow; when a flow sends a burst of packets at a rate greater than its reserved rate, extra packets are held until their eligibility time (determined based on the reserved rate for the flow), even if idle bandwidth is available for transmitting these packets. A CJVC network, therefore, does not meet our design requirement of utilizing resources efficiently.

In this dissertation, we design work-conserving core-stateless networks that provide deterministic per-flow guarantees, similar to core-stateful networks, on end-to-end delay[3], throughput, and fairness.

Research Methodology Our approach blends theory and practice. We address the theoretical aspects of designing core-stateless networks in two steps: (1) we understand the end-to-end service guarantees that can be provided in core-stateful networks; and (2) we design core-stateless networks that provide end-to-end service guarantees similar to core-stateful networks. On the practice front, we design and implement a router prototype to evaluate the feasibility of deploying our core-stateless architecture, and investigate the scalability of routers. We summarize our contributions below.

End-to-end analysis of core-stateful networks

Past Work Past analysis of Guaranteed Rate (GR) [25] networks derives the end-to-end *delay* guarantees that can be provided using a core-stateful network architecture. The *Packet Scale Rate Guarantee* (PSRG) [5] framework helps to understand the end-to-end *throughput* guarantees provided by a fair queuing network. However, there are no analyses that derive the end-to-end fairness properties of a core-stateful network of fair servers.

Our Contribution We analyze networks that employ fair scheduling algorithms to understand the end-to-end *fairness* guarantees that they can provide. Our analysis results in the *first* tight bound on end-to-end fairness for any network of fair servers. We first argue that it is difficult to extend existing single-node fairness analysis to an end-to-end analysis of a network where each node may employ a dif-

[3]Recently, and simultaneous to our work, work-conserving core-stateless networks that provide delay guarantees have been proposed in [40, 66].

ferent fair scheduling algorithm. We then present a two-step approach for end-to-end fairness analysis of heterogeneous networks. First, we define a class of scheduling algorithms, referred to as the *Fair Throughput* (FT) class, and prove that most known fair scheduling algorithms belong to this class. Second, we develop an analysis methodology for deriving the end-to-end fairness bounds for a network of FT servers. Our analysis is general and can be applied to heterogeneous networks where different nodes employ different scheduling algorithms from the FT class.

Designing core-stateless architectures

CSGR Networks We design a methodology to transform any core-stateful network that employs scheduling algorithms from the GR class, to its core-stateless version (CSGR) that provides the same end-to-end delay guarantee. The key insight we use is that: *upper bounds on packet deadlines at any core node can be computed using per-flow state at the edge node.* We demonstrate that a CSGR network, that uses the upper bounds on deadlines, instead of actual deadlines, provides the same end-to-end delay guarantee. Since the GR class is fairly general, this methodology provides a tool to design a wide range of core-stateless networks that provide delay guarantees. For instance, it is possible to design a core-stateless Delay-EDD network, that decouples the delay and rate guarantee.

CSGT Networks We propose the Core-stateless Guaranteed Throughput (CSGT) network architecture—the *first* work-conserving network architecture that provides throughput guarantees to individual flows over finite time-scales, but without maintaining per-flow state in core routers. We develop the architecture in two steps. First, we show that for a network to provide end-to-end throughput guarantees, it must also provide end-to-end delay guarantees. Second, we demonstrate that two mechanisms —tag re-use and source rate control— when integrated with a work-conserving, core-stateless network that provides end-to-end delay guarantees, lead to the design of CSGT network that provides end-to-end throughput bounds within an *additive constant* of what is attained by a core-stateful network of fair rate servers. We demonstrate that the constant is small for current network topologies.

Providing Fairness Guarantees in Core-stateless Networks We propose the Core-stateless Guaranteed Fair (CSGF) network architecture—the *first* work-conserving core-stateless architecture that provides deterministic fairness guarantees. We develop the architecture in two steps. First, we show that for a network to provide fairness guarantees, it must also provide throughput guarantees. Second, we demonstrate that a set of two mechanisms—fair access at the edge and aggregation of flows in the core—when integrated with a CSGT network that provides throughput guarantees, lead to the design of CSGF networks that provide fairness guarantees. The fairness guarantees provided by a CSGF network on application throughput are comparable to those provided by core-stateful networks.

Scalability Evaluation on a Programmable Router Platform

We have shown that our core-stateless architectures come close to their core-stateful counterparts in providing per-flow guarantees. We next evaluate the scalability of routers in our architectures—and compare them to routers in FIFO and IntServ networks—by implementing them on a programmable router platform. Our results indicate that:

1. Core routers in core-stateless networks that employ source routing can support best-case packet-processing speeds similar to those in conventional FIFO IP networks; they halve the gap between the worst-case packet-processing speeds of routers in FIFO and IntServ networks.

2. The memory space requirement of core routers in a core-stateless architecture is similar to that of routers in conventional FIFO networks.

The rest of this dissertation is organized as follows. We present the evaluation of the performance of CBR flows in FIFO networks in Chapter 2. We present the end-to-end analysis of core-stateful networks in Chapter 3. The design of core-stateless networks that provide delay, throughput, and fairness guarantees, is presented in Chapters 4, 5, and 6, respectively. The scalability evaluation of the core-stateless architectures on a programmable router platform is presented in Chapter 7. Chapter 8 presents the conclusions of this dissertation and directions for future research.

1.6 Notation and Assumptions

Throughout this dissertation, we use the following symbols and notations.

p_f^k	:	the k^{th} packet of flow f
$a_{f,j}^k$	:	arrival time of p_f^k at node j on its path
$d_{f,j}^k$	:	departure time of p_f^k from node j
l_f^k	:	length of packet p_f^k
r_f	:	rate reserved for flow f
π_j	:	upper bound on propagation delay of the link connecting node j and $(j+1)$
C_j	:	outgoing link capacity at node j

We use H to denote the number of routers along the path of flow f; the source of flow f is connected to router 1 and the destination is connected to router H. The k^{th} packet transmitted from the source, p_f^k, is said to have a *sequence number* of k. A source is said to transmit packets *at least at its reserved rate*, if $a_{f,1}^k \leq a_{f,1}^{k-1} + \frac{l_f^{k-1}}{r_f}$. We use the terms *server* and *router* interchangeably; further, we make the assumption that the sum of rates reserved for flows at any server does not exceed the server capacity (i.e., the link bandwidth).

Chapter 2

Performance of CBR Flows in FIFO Networks

Providing delay guarantees is fundamental to providing any other types of network service guarantees. For instance, it can be shown that a network that does not provide delay guarantees cannot provide throughput, fairness, or jitter guarantees. It is well known that FIFO networks do not provide delay guarantees; a burst of packet arrivals from a flow results in large queuing delays for all other flows sharing the network. The natural question is: *can FIFO networks provide a low-delay service if bursts are not allowed into the network?* In fact, the conjecture that preventing bursts from entering a FIFO network is sufficient to provide a low-delay and no-loss service is at the basis of the design of the well-known Virtual Leased Line (VLL) service model [11, 31, 44]. In this chapter, we evaluate the validity of this conjecture.

2.1 Background

Evaluation of the performance of CBR flows in FIFO networks has generated interest in the past. In [4], it is shown that given any level of network utilization, worst-case examples of network topologies and input traffic can be constructed, such that packets experience unbounded delay even when flows are shaped to CBR at the source [4]. This demonstrates that CBR flows can become bursty as they traverse a FIFO network. These examples, however, are artificial in nature, and do not lend insights into the delay and jitter performance of CBR flows in realistic FIFO networks.

Delay Bounds in FIFO Networks Worst-case bounds on end-to-end delay experienced by CBR flows in FIFO networks have been provided under special operating conditions in [4, 12, 39]. The bounds derived in [4] are valid only for networks that operate at utilization lower than $\frac{1}{H-1}$, where H is the maximum number of hops on an end-to-end path. The bounds derived in [12, 39] are valid when the bit-rates of CBR sources are constrained by the total number of flows that share any router with them on the end-to-end path. These delay bounds are, therefore, not applicable to general and realistic FIFO networks.

Past Analytical Work A number of analytical models have been proposed for characterizing the delay and jitter performance of flows in general FIFO networks [19, 28, 42, 48, 50, 55].

Bounds on the queue size distribution for superposed CBR streams with heterogeneity in their periods have been derived in [50]. Techniques for estimating the end-to-end jitter incurred to CBR traffic in an ATM network have been provided in [42]. Their study is, however, limited to flows with fixed packet sizes, and does not investigate the actual end-to-end delay incurred. A connection traversing multiple hops has been studied in [28] and it has been observed that the end-to-end delay distribution depends on the auto-covariance of the cross-traffic sharing the network links. It has been shown in [19] that the inter-arrival time of packets becomes more bursty in a traffic stream on passing through networks that carry packets of highly variable size. However, this study focuses on non-CBR traffic, and it is not obvious if the same result would hold for CBR networks. Furthermore, the analysis is carried out assuming a single cross traffic stream and a small network. In [48], the authors model CBR flows with an aggregated CBR background traffic at a single network node, and extend it to the end-to-end case in [55] under high utilization. The main inadequacy of these models is that these are very complicated, computationally intensive, and do not yield any closed-form characterization of, or direct insights into the performance of CBR flows.

Past Experimental Work A simulation study that directly measures the end-to-end delay and jitter observed by CBR flows as they traverse large networks while interacting with several cross traffic flows has been presented in [27]. However, their study assumes fixed packet sizes, making the study applicable only to ATM-type networks.

Our Approach We address the limitations of past analytical and simulation studies on the performance of CBR flows in FIFO networks in two steps: (1) we develop an analytical model that yields a closed-form characterization of the end-to-end performance of CBR flows in FIFO networks under *asymptotic* conditions of network utilization and path length; and (2) we conduct simulations to verify the set of *non-asymptotic* and realistic conditions under which the results of the model continue to hold.

We present these in some detail in the rest of this chapter.

2.2 Analytical Model

We have developed an analytical model for the inter-arrival time between packets at the destination for individual flows (including CBR flows) under *asymptotic* conditions of network utilization and number of hops traversed [60].

Our analysis proceeds in the following basic steps. First, it relates the variance in inter-arrival time of packets of a given flow at node j to that at node $j-1$. Next, it shows that the variance of the limiting distribution of inter-arrival times is unbounded. It establishes that the mean of the limiting distribution, however, is finite. Finally, it shows that a distribution with a finite mean, but unbounded variance, is *heavy-tailed*[1]. In this section, we summarize the insights gained from our analysis; the details of the model can be found in [60].

Our model yields the following key insights:

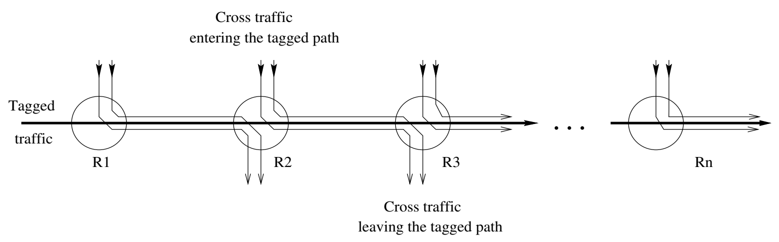
1. The variance in inter-arrival times for individual flows tends to infinity in the asymptotic case, indicating that flows become *heavy-tailed* after traversing a network of FIFO routers.

2. The aggregation of flows becomes a long-range dependent self-similar process.

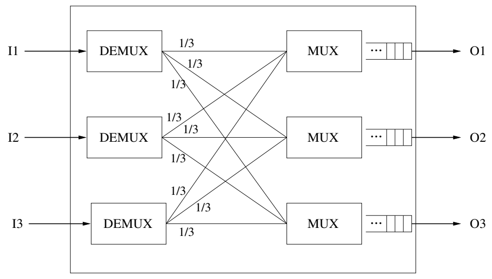
The occurrence of heavy-tails in the inter-arrival times of flows implies that a non-negligible fraction of packets experience significantly large delays. Since the model results are valid even for CBR flows, it follows that shaping flows to CBR at the edge of the network does not result in a low-delay performance in networks that operate at high levels of utilization.

Observe that the analytical model predicts heavy-tails in flows when the network utilization and path length are asymptotically high: 100% and infinite, respectively. However, such conditions do not arise in practice. Hence, it is not clear from the analysis alone, whether CBR flows become heavy-tailed even in realistic networks. To address this question, we conduct experimental studies and measure the delay behavior of CBR flows under *non-asymptotic* network and traffic conditions. We describe these studies next.

[1]A heavy-tailed distribution is one in which the probability of taking very large values is non-negligible.



(a) Network Topology



(b) Router Architecture

Figure 2.1: Simulation environment: network topology and router architecture

2.3 Experimental Evaluation

We have developed a network simulator using CSIM [1] to study the end-to-end performance of CBR flows in large-scale FIFO networks. In this section, we describe our simulation environment and the metrics for the performance evaluation.

2.3.1 Simulation Environment

Network Topology

For our experiments, we consider a linear, multi-hop network topology (see Figure 2.1(a)). This network model is fairly general and has been used in literature [28, 38, 42, 62]. Let M_n denote a linear, multi-hop network topology with n routers, and let R_i ($i \in [1, n]$) denote the ith router in the topology. Given such a topology, we are interested in the end-to-end performance of the *tagged traffic*, which refers to the set of CBR flows that enters the network topology at router R_1 and traverses the multi-hop network topology M_n. Specifically, we measure how the characteristics of the individual CBR flows aggregated in the tagged traffic are altered as they interact with other CBR aggregates (referred to as the *cross traffic*) that enter and depart the network at each router along the path.

We model each router in this network as having p input ports ($I_1, \ldots, I_p$) and p output ports ($O_1, \ldots, O_p$). The network topology M_n consists of n routers such that, for all i ($1 \leq i \leq n-1$), the output port O_1 of router R_i is connected to the input port I_1 of router R_{i+1}. A tagged flow enters the network through port I_1 of router R_1. Through each port $I_2, \ldots, I_p$ of router R_i, aggregates enter the path, and $1/p$ of each of these aggregates are routed to output port O_1 (see Figure 2.1(b)). In addition, the tagged traffic entering input port I_1 of R_i is routed to output port O_1. Thus, for each router, the traffic routed to the output port O_1 consists of: (1) The tagged traffic (entering the router from port I_1); and (2) $1/p$ of the flows entering from input ports $I_2, \ldots, I_p$. All of the remaining traffic entering each router is routed to output ports $O_2, \ldots, O_p$.

The above topology ensures that: (1) the tagged traffic that enters the network at router R_1 is routed all the way through the multi-hop network M_n, and (2) the cross traffic entering the network at router R_i ($i \in [1, n]$) interferes with the transmission of the tagged traffic for a single hop, and leaves the network at router R_{i+1}. This topology facilitates experimentation with different compositions of the cross traffic and different network depths. We have conducted experiments for p ranging from 8 to 32. We present results for experiments with $p = 8$; these results hold for other values of p too.

Modeling Cross Traffic

The extent to which the cross traffic entering each router affects the characteristics of the CBR flows in the tagged traffic depends on the *burstiness* of the cross traffic. Note that, although we have assumed that each flow entering the network is shaped to CBR at the source, the aggregate cross traffic entering each router may be bursty. This burstiness results from: (1) the heterogeneity in the inter-arrival times of the CBR flows, and (2) the traffic distortions that result from flows traversing through multiple routers in the network.

To reasonably approximate traffic distortions, we model the cross traffic entering at each router in the network as consisting of two types of flows: (1) flows that are at the beginning of their routes or have traversed through a *small* (M_1 or M_2) number of routers, and (2) flows that are at the end of their routes or have traversed through a *large* (M_{20}) number of hops. This model closely approximates the current Internet—each backbone router is a small number of hops away from some set of hosts while being far away from some others.

To capture heterogeneity across CBR flows, we consider two classes of cross-traffic flows—flows with *large* and *small* average packet inter-arrival times at the source. We quantify the heterogeneity in cross-traffic flows in terms of the *inter-arrival time ratio* (IATR), which is defined as the ratio of the average packet inter-arrival times for these two classes of flows. We construct the two flow classes in the following two ways:

1. *CBR flows with heterogeneous packet sizes:* All CBR flows have the same bit-rate requirement, but their packet size is selected uniformly from two intervals.

2. *CBR flows with heterogeneous bit-rates:* All CBR flows have the same packet size, but their bit-rate requirement is selected uniformly from two intervals.

2.3.2 Measuring Non-asymptotic Region of Heavy-tailed Behavior for CBR Flows

The analytical model in [60] predicts a heavy-tailed behavior for CBR flows under asymptotically high utilization and path length. In this section, our objective is to compute—through simulations—the *non-asymptotic* set of network and traffic conditions, under which CBR flows become heavy-tailed.

Tests Used to Detect Heavy-tailed Behavior in CBR Flows The literature contains the following tests for identifying heavy-tailedness in flows:

1. LLCD Test: The log-log complimentary (LLCD) test is used to detect the presence of heavy-tails in a flow. The complimentary distribution of the packet inter-arrival times is plotted on a log-log scale. For a heavy-tailed flow with a finite mean inter-arrival times, the distribution looks like a straight line with slope $\alpha \in (1, 2)$, for large values of inter-arrival times.

2. *Hill Test:* Let $U_1, \ldots, U_n$ denote the packet inter-arrival times of a given flow in ascending order. The Hill function, $hill(k)$, for a flow is defined as:

$$hill(k) = \left[\sum_{i=0}^{k-1} k^{-1} \log \frac{U_{n-i}}{U_{n-k}}\right]^{-1}$$

   A source is said to be heavy-tailed with finite mean if $hill(k)$ stabilizes to $\alpha \in (1, 2)$ for large value of k.

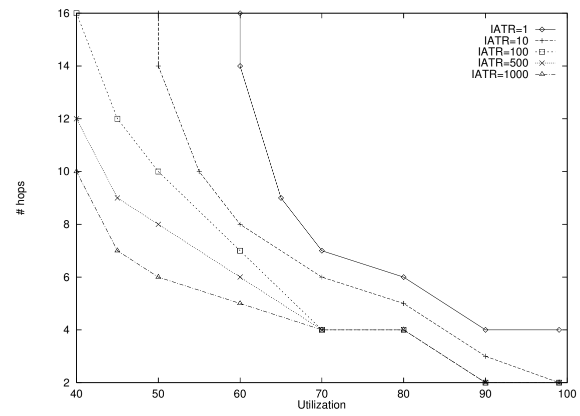
Parameter Settings We simulate an M_{20} topology with $40Mbps$ links. We select a tagged flow with a bit-rate requirement of $4Mbps$, and packets of size $100B$.

We use the following *dimensions* to identify the non-asymptotic *region of applicability* of the model:

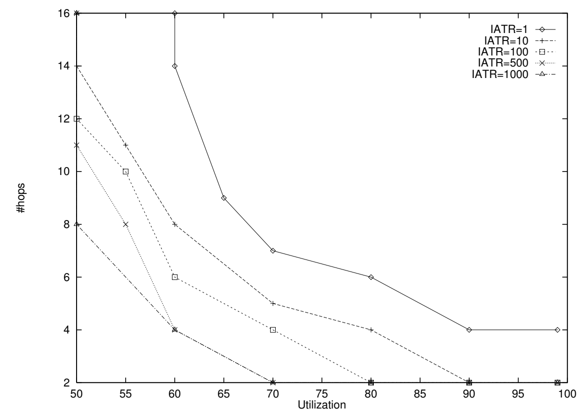
- **Network Utilization:** To identify the levels of network utilization above which CBR flows exhibit heavy-tailed behavior, we conduct experiments with link utilization varying from 20% to 99%.

- **Path Length:** The greater the number of hops a flow traverses, the larger the likelihood of its inter-arrival time becoming bursty. We evaluate the effect of increasing the number of hops—from 2 to 20—on the validity of the model.

- **IATR:** The greater the heterogeneity in the cross-traffic, the greater is its interference with the tagged flow. We evaluate the effect of this parameter by varying the IATR of the cross-traffic from 1 to 1000.



(a) CBR: Heterogeneous packet sizes



(b) CBR: Heterogeneous bit rates

Figure 2.2: Region of applicability of the analytical model

We conduct experiments with network settings defined by a combination of specific values for utilization, network depth, and IATR. Any network setting that yields a heavy-tailed tagged flow (with $\alpha \in (1, 2)$) at the destination is said to belong to the *region of applicability* of the model.

Experimental Results For the two different cross-traffic settings, Figures 2.2(a) and 2.2(b) plot the *minimum* utilization level versus the *minimum* number of router hops at that utilization necessary for observing $\alpha \in (1, 2)$ for the tagged flow. In each of these experiments, the tagged flow occupies 10% of the total traffic sharing a link. For each value of IATR, the area above and to the right of its corresponding curve identifies the *region of applicability* of the model. The following conclusions can be drawn from these figures:

1. The greater the heterogeneity in the cross-traffic, the larger is the region of applicability of the model (i.e., tagged flows show heavy-tailed behavior for a larger set of network settings).
2. For same value of IATR, heterogeneity in packet-sizes results in a slightly larger *region of applicability* of the model than heterogeneity in bit-rates.

3. More importantly, these figures demonstrate that flows can become heavy-tailed in large heterogeneous networks running at relatively low utilization levels (e.g., 40%), or after traversing as few as 4 hops in networks running at high utilization levels.

These simulations demonstrate that the end-to-end delay performance of CBR flows is not adequate for applications with timeliness requirements in large-scale FIFO networks operating at moderate to high levels of utilization. The use of per-flow scheduling algorithms has been widely suggested in the literature to provide good end-to-end delay performance [18, 64]. In the following, we compare the performance of CBR flows in FIFO networks to that in networks that employ per-flow queuing mechanisms.

2.3.3 Comparing Performance of CBR Flows in FIFO Networks vs. Per-flow Queuing Networks

Background To address the limitations of FIFO, several different packet scheduling algorithms have been proposed [6, 18, 26, 64]. These algorithms maintain per-flow state and provide bounded end-to-end delay guarantee to a flow regardless of the behavior of other flows in the network. It can be shown that, for a CBR flow, the maximum end-to-end delay in a typical network that employs per-flow scheduling algorithms, is bounded by [25, 52]:

$$K * I_f + \sum_{j=1}^{j=K} \beta_j \qquad (2.1)$$

where $I_f = \frac{l_f}{r_f}$ is the packet inter-arrival time of the CBR flow at the source, and β_j is a constant that depends on the scheduling algorithm at switch j. Additionally, the inter-arrival time at the *destination* is bounded by [23]:

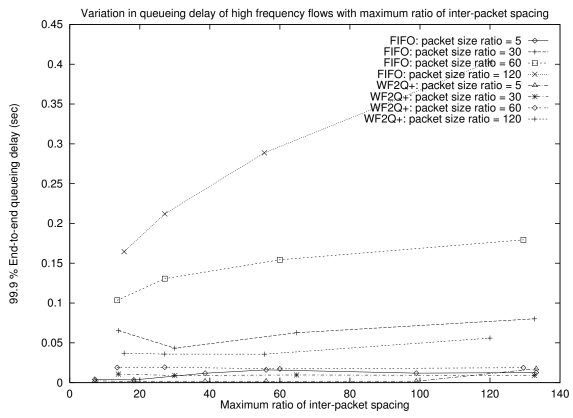
$$K * I_f + \sum_{j=1}^{j=K} \alpha_j \qquad (2.2)$$

where α_j is a constant that depends on the scheduling algorithm at switch j. When normalized by I_f, both—the end-to-end delay and the packet inter-arrival time at the destination— are bounded by the number of nodes on the end-to-end path (plus a constant). Our objective is to compare the maximum delay experienced by CBR flows in FIFO networks, to that in per-flow queuing networks that guarantee the above bounds on worst-case performance.

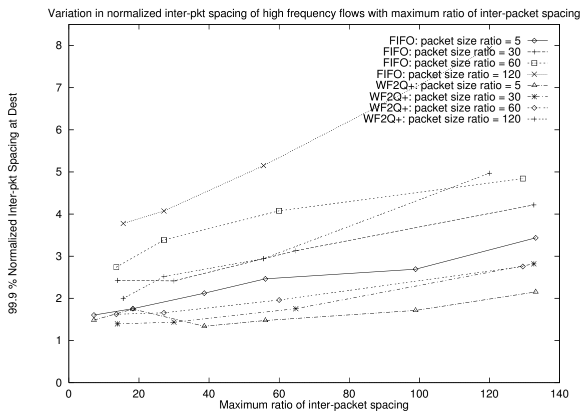
Experimental Design We conduct experiments to compare the worst-case end-to-end performance of CBR flows in networks where each router employs either FIFO or per-flow queuing to arbitrate access to link bandwidth, and all flows are shaped to CBR at the source or at the ingress routers. We use WF^{2}Q+ [7] as a representative per-flow scheduling algorithm. We experiment with different settings of network utilization, path length, and heterogeneity among CBR flows (IATR). In this chapter, we present results only for path lengths of 20; the detailed results can be found in [52]. The link capacities are set to 40Mbps.

Heterogeneity Across CBR Flows We investigate the effect of heterogeneity in *both* the bit-rate requirements and the packet sizes across CBR flows, on their end-to-end performance. To conduct this experiment, we simulate a series of network environments with different ratios of packet sizes (namely, 5, 30, 60, and 120) for the two classes of CBR flows. For each environment, we select the bit-rate requirement of flows such that the IATR varies from 15 to 130. Each network is operated at a utilization of 97%. Figure 2.3 depicts the results of these experiments for the class of CBR flows with high packet frequency.

Figure 2.3(a) plots the 99.9%-percentile of the distributions of the end-to-end queuing delay as a function of the IATR. Each line represents cross-traffic scenarios with the same ratio of packet sizes for the two classes of CBR flows. The graph indicates the following. (1) Increasing heterogeneity in packet sizes increases the



(a)



(b)

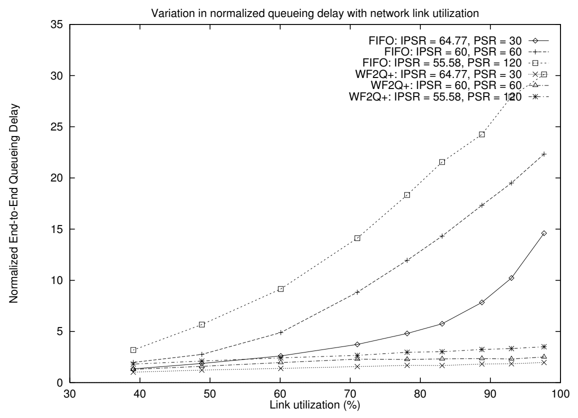
Figure 2.3: 99.9%-percentile of (a) end-to-end queuing delay and (b) normalized inter-arrival time (for high packet frequency flows)

maximum end-to-end queuing delay. The maximum delay in a FIFO network can be an order of magnitude larger than the end-to-end propagation latency ($10ms$). (2) For networks that support CBR flows with small heterogeneity in packet sizes, increasing the ratio of packet arrival rates (by changing the ratio of bit-rate requirements of flow classes) does not yield any noticeable increase in the end-to-end queuing delay. However, for network environments that support larger heterogeneity in packet sizes (e.g., packet size ratios of 60 and 120), increasing the ratio of packet arrival rates increases the maximum end-to-end delay suffered by flows with high arrival rates.

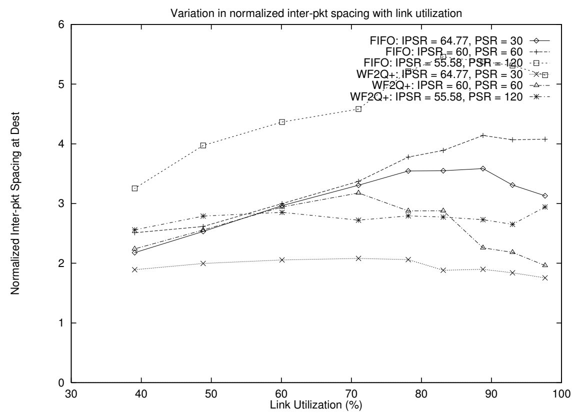
Figure 2.3(b) supports that same conclusion with respect to the normalized inter-packet separation.

Effect of Network Utilization The previous experiments are conducting on networks operating at 97% utilization. We next evaluate the effect of different network utilization levels (ranging from 40% to 97%) on the results. Figure 2.4 plots the results of this experiment.

Figure 2.4(a) indicates that for flows with high packet arrival rates, with



(a)



(b)

Figure 2.4: 99.9%-percentile of (a) normalized end-to-end queuing delay and (b) normalized inter-arrival time

FIFO scheduling, the end-to-end queuing delay reduces by a factor of 10 as the network utilization decreases from 97% to 40%. To compare the delay in FIFO networks with the bounds guaranteed by per-flow queuing networks, we normalize it with respect to I_f, the inter-arrival time at the source. We find that FIFO delays can not be bounded in terms of the intrinsic properties of a flow. In the presence of heterogeneous CBR flows, FIFO yields higher end-to-end queuing delays than WF^{2}Q+ even at $40 - 50\%$ utilization.

Figure 2.4(b) illustrates that normalized inter-packet spacing at the destination is relatively independent of the network utilization.

From the above experiments, we conclude that the difference in performance of FIFO and fair queuing networks is significant in environments with heterogeneous packet sizes (like the current Internet), and those that operate at moderate to high levels of utilization (greater than around 50%).

2.4 Discussion

From the above studies, we conclude that FIFO scheduling is not sufficient to provide a low-delay network service when CBR flows occupy moderate-to-large fractions of available capacity. The reason for this is that bursts can accumulate even within CBR flows as they traverse a network. These effects are more pronounced at higher levels of network utilization. Thus, to provide a low-delay and no-loss service to CBR flows, a FIFO network has to either limit the network utilization as seen by CBR flows, or add mechanisms to prevent the accumulation of bursts. We discuss both of these approaches:

1. A FIFO network can limit the network utilization seen by CBR flows (1) by ensuring that the cumulative bit-rate of the CBR flows is a small fraction of the total capacity, and (2) by assigning higher-priority to this class of flows. Traffic classes that are assigned lower priority, for instance, best-effort traffic, can utilize the remainder of the network capacity. This approach may be sufficient if the network offers only one type of premium service; for a FIFO network to offer multiple types of premium service classes, such as a guaranteed throughput class, however, the above approach requires that the cumulative bit-rate of flows from *all* of these classes be limited to a small fraction of the total capacity. FIFO scheduling is therefore not adequate to design networks that devote a significant fraction of their resources to support customers that require some kind of absolute performance guarantees.

2. Accumulation of traffic bursts within the network may be eliminated by employing CBR shapers at *all* routers. However, such an approach requires maintenance of per-flow timers in addition to per-flow state, and requires packet classification at core routers. As argued before, these requirements threaten the scalability of core routers in high-performance networks. Further, per-node

CBR shapers render the network non work-conserving, which is undesirable.

Conclusion 1 *A FIFO network can provide a low-delay network service by (1) shaping flows that subscribe to this service class to CBR, (2) assigning higher-priority to this service class, and (3) ensuring that the cumulative bandwidth occupied by flows in this class is a small fraction of the total capacity. FIFO scheduling is inadequate to design scalable networks that devote a significant fraction of available resources to support customers that require per-flow service guarantees.*

This chapter concludes our investigation of the performance of CBR flows in FIFO networks. In the rest of this dissertation, we explore our second research direction of designing work-conserving core-stateless network architectures that provide per-flow service guarantees.

Chapter 3

End-to-end Service Guarantees in Core-stateful Networks

The need to provide per-flow service guarantees in networks has been recognized for some time. In the last decade, several sophisticated packet scheduling algorithms, that provide *per-hop* guarantees with respect to delay, throughput, fairness, and jitter, have been proposed [7, 22, 26, 65]. To provide such guarantees, however, these algorithms need to maintain per-flow state. As a measure against which to evaluate the service guarantees provided by core-stateless network architectures that do not maintain state in the core, it is important to understand the *end-to-end* guarantees provided by core-stateful networks.

In general, the end-to-end analysis of a network of servers, each of which provides a *per-hop* service guarantee, is not straightforward. This is due to two main reasons:

1. The literature contains single-server performance analyses only for *specific* scheduling algorithms [6, 7, 22, 26]. In a wide-area network, however, each router may employ a different scheduling algorithm. Hence, an end-to-end analysis should be applicable to such heterogeneous networks.

2. Most single-server analyses of scheduling algorithms assume certain input traffic models. Due to the variability in the delay experienced by packets at a server, traffic gets distorted as it traverses the network. Therefore, input traffic models that enable analysis at the first server in a network, may not remain valid at subsequent nodes. Hence, it is not straightforward to extend existing single-server analyses of scheduling algorithms to end-to-end analyses.

In this chapter, we build upon past work to understand the end-to-end delay and throughput guarantees that can be provided in core-stateful networks, and present a novel analysis to derive end-to-end fairness guarantees.

3.1 End-to-end Delay Guarantees

Single-server Delay Guarantees

The *expected arrival time* of packet p_f^k of flow f at server j is defined as [64]:

$$\begin{aligned} EAT_j(p_f^1) &= a_{f,j}^1 \\ EAT_j(p_f^k) &= \max\left(a_{f,j}^k, EAT_j(p_f^{k-1})\right) + \frac{l_f^{k-1}}{r_f}, \ k > 1 \end{aligned}$$

Then, server j is said to provide a *delay guarantee* to flow f, if it provides an upper bound on $(d_{f,j}^k - EAT_j(p_f^k))$, the difference between the departure time of p_f^k from the server and its expected arrival time at the server [61]. Note that given a source traffic characterization, such as leaky-bucket, a delay guarantee can be used to derive upper bounds on the delay suffered by packets of a flow at server j [25].

Most per-flow scheduling algorithms proposed in the literature provide delay guarantees. In [25], the *Guaranteed Rate* (GR) framework has been defined to characterize such algorithms. The formal definition is as follows:

Definition 1 *A scheduling algorithm at server j belongs to class GR for flow f if it guarantees that packet p_f^k will be transmitted by $GRC_{f,j}^k + \beta_{f,j}$, where $GRC_{f,j}^k =$*

GR algorithm	$\beta_{f,j}$
Virtual Clock	l_j^{max}/C_j
Packet-by-Packet GPS	l_j^{max}/C_j
Self Clocked Fair Queuing	$\sum_{m \neq f} l_j^{max}/C_j$
Delay EDD	$l_j^{max}/C_j + d_{f,j} - l_f/r_{f,j}$

Table 3.1: $\beta_{f,j}$ values of some GR algorithms (For Delay EDD, $d_{f,j}$ is the desired delay bound for flow f at server j)

$EAT_j(p_f^k) + \frac{l_f^k}{r_f}$, *and* $\beta_{f,j}$ *is a constant which depends on the scheduling algorithm and the server.*

From the definition, it follows that the delay guarantee of a GR server is given by: $d_{f,j}^k - EAT_j(p_f^k) \leq \frac{l_f^k}{r_f} + \beta_{f,j}$. Table 3.1 lists the values of $\beta_{f,j}$ for some GR algorithms (derived in [25]).

Deriving End-to-end Delay Guarantees

Consider a flow f that traverses a path of H servers through a network. The network is said to provide an *end-to-end delay guarantee* if it provides an upper bound on $(d_{f,H}^k - EAT_1(p_f^k))$, the difference between the departure time of the packet from the network and its expected arrival time into the network (at the *first* server).

An end-to-end analysis of a network of GR servers has been presented in [24], which derives the following end-to-end delay guarantee:

Theorem 1 *If the scheduling algorithm at each of the servers on the path of a flow belongs to GR for flow f, then the departure time of packet p_f^k from the last node in the network is given by:*

$$d_{f,H}^k \leq EAT_1(p_f^k) + \frac{l_f^k}{r_f} + (K-1) \max_{i \in [1..k]} \frac{l_f^i}{r_f^i} + \sum_{j=1}^{H} (\beta_{f,j} + \pi_j)$$

Past work on the end-to-end analysis of GR networks, thus, helps us understand the end-to-end delay guarantees that can be provided using a core-stateful network architecture.

3.2 End-to-end Throughput Guarantees

Single-server Throughput Guarantees

The throughput of a flow in a time interval is defined as the number of bits belonging to that flow delivered by a server (or a network) during that time interval. A flow with a reserved rate of r_f at a server, would expect the server to provide it throughput, during any time interval, at least at the rate r_f. To capture such a service semantics, we define a single-server *throughput guarantee* as follows:

Definition 2 *A server is said to provide a throughput guarantee to a flow f, whose source transmits packets at least at its reserved rate, if in any time interval $[t_1, t_2]$, the server guarantees a minimum throughput to flow f, $W_{f,j}(t_1, t_2)$, given by:*

$$W_{f,j}(t_1, t_2) > r_f(t_2 - t_1) - r_f\ \gamma_{f,j} \tag{3.1}$$

where $\gamma_{f,j}$ is a constant that depends on the traffic and server characteristics.

From the definition, a server guarantees a non-zero throughput to flow f if $t_2 - t_1 > \gamma^{net}_{f,H}$; thus, the value of $\gamma^{net}_{f,H}$ bounds the longest time interval for which a flow may receive no throughput from the server. Clearly, the smaller the value of $\gamma^{net}_{f,H}$, the better the quality of service for applications that require sustained throughput.

Observe that most scheduling algorithms that provide delay guarantees, also guarantee an *average* throughput at the reserved rate; however, these algorithms differ in the time-scales (namely, the value of $\gamma^{net}_{f,H}$) at which this guarantee is provided. For instance, at a server that employs an unfair packet scheduling algorithm (e.g., Virtual Clock or Delay EDD), the throughput received by a flow during a time interval is a function of the throughput received by the flow in the past. In fact, for such servers, $\gamma_{f,j}$ is not bounded, indicating that an unfair server cannot guarantee non-zero throughput at finite time scales.

It has been shown that most fair queuing algorithms provide single-server throughput guarantees [6, 22, 26]. The *Packet Scale Rate Guarantee* (PSRG) framework proposed recently is useful to capture the throughput properties of such algorithms [5]. We recall the definition below.

Definition 3 *A server j is said to provide a "packet scale rate guarantee r_f with latency $\epsilon_{f,j}$" to a flow f if the departure time of packet p_f^k satisfies [5]:*

$$d_{f,j}^k \leq F_j(p_f^k) + \epsilon_{f,j}$$

where F_j is recursively defined as:

$$\begin{aligned} F_j(p_f^1) &= a_{f,j}^1 + \frac{l_f^1}{r_f} \\ F_j(p_f^k) &= max\left(a_{f,j}^k, min(d_{f,j}^{k-1}, F_j(p_f^{k-1}))\right) + \frac{l_f^k}{r_f}, \qquad k > 1 \end{aligned} \tag{3.2}$$

Fair scheduling algorithms such as PGPS [47] and WF^{2}Q [6] have been shown to provide the packet scale rate guarantee.

Lemma 1 shows that a server that provides a packet scale rate guarantee, also provides a single-server throughput guarantee.

Lemma 1 *A server j that provides a "packet scale rate guarantee r_f with latency $\epsilon_{f,j}$" to a flow f, the source of which transmits at least at its reserved rate r_f, also provides a minimum throughput to flow f, during any time interval $[t_1, t_2]$, given by: $r(t_2 - t_1) - r_f\ \epsilon_{f,j} - 2l_f^{max}$, where l_f^{max} is the largest packet size used by flow f.*

Proof: Observe that: $F_j(p_f^{k-1}) \geq a_{f,j}^{k-1} + \frac{l_f^{k-1}}{r_f}$. Also observe that if the source of flow f transmits at least at its reserved rate, then $a_{f,j}^k \leq a_{f,j}^{k-1} + \frac{l_f^{k-1}}{r_f} \leq F_j(p_f^{k-1})$.

Using this fact and decomposing the *max* term in (3.2), it can be observed that for such a source:

$$F_j(p_f^k) \leq F_j(p_f^{k-1}) + \frac{l_f^k}{r_f} \tag{3.3}$$

Consider any time interval $[t_1, t_2]$. Consider the following two cases:

- *There is no backlog of packets of flow f at t_1.*

  Let p_f^k be the first packet to arrive in the interval $[t_1, t_2]$. Then $a_{f,j}^{k-1} < t_1$, which implies that $a_{f,j}^k < t_1 + \frac{l_f^{k-1}}{r_f}$. Therefore, $F_j(p_f^k) = a_{f,j}^k + \frac{l_f^k}{r_f} < t_1 + \frac{l_f^{k-1}}{r_f} + \frac{l_f^k}{r_f}$.

- *There is a backlog of packets of flow f at t_1.*

  Let p_f^k be the packet of flow f with the smallest F_j value in the backlog. Then $d_{f,1}^{k-1} < t_1$ (by definition of p_f^k), and $a_{f,j}^k < t_1$ (because p_f^k is backlogged). From (3.2), it can therefore be observed that $F_j(p_f^k) < t_1 + \frac{l_f^k}{r_f}$.

Next observe from the definition of packet scale rate guarantee, that any packet assigned an F_j value no more than $t_2 - \epsilon_{f,j}$, will depart the server by t_2. This implies that among the packets that depart the server after t_1, those with $F_j \in [t_1, t_2 - \epsilon_{f,j}]$, will depart the server in the time interval $[t_1, t_2]$. From the two cases above, if p_f^k is the packet with the smallest F_j value to depart after t_1, then $F_j(p_f^k) < t_1 + \frac{l_f^k}{r_f} + \frac{l_f^{k-1}}{r_f} \leq t_1 + \frac{l_f^k}{r_f} + \frac{l_f^{max}}{r_f}$. From (3.3), therefore, it can be seen that among packets that depart after t_1, packets with a total of at least $r_f(t_2 - t_1 - \epsilon_{f,j}) - 2l_f^{max}$ bits are assigned F_j values that lie in the interval $[t_1, t_2 - \epsilon_{f,j}]$—the second l_f^{max} term appears because the F_j value of the last of this sequence of packets could lie anywhere in $(t_2 - \epsilon_{f,j} - \frac{l_f^{max}}{r_f}, t_2 - \epsilon_{f,j}]$. Therefore, the throughput received by flow f during the time interval $[t_1, t_2]$ is at least $r(t_2 - t_1) - r_f\ \epsilon_{f,j} - 2l_f^{max}$. ■

Deriving End-to-end Throughput Guarantees

A flow that reserves a rate of r_f at all servers in its path, would expect the network to provide it throughput at least at the rate r_f. This is captured by an *end-to-end* *throughput guarantee* defined below.

Definition 4 *A network is said to provide a throughput guarantee to a flow f, whose source transmits packets at least at its reserved rate, if in any time interval $[t_1, t_2]$, the network guarantees a minimum throughput to flow f, $W^{net}_{f,H}(t_1, t_2)$, given by:*

$$W^{net}_{f,H}(t_1, t_2) > r_f(t_2 - t_1) - r_f \ \gamma^{net}_{f,H} \tag{3.4}$$

where $\gamma^{net}_{f,H}$ is a constant that depends on the traffic and server characteristics at different nodes on the end-to-end path.

An end-to-end Packet Scale Rate Guarantee for a network of servers has been derived in [5][1]. Using Lemma 1, a corresponding end-to-end throughput guarantee can be stated as:

Theorem 2 *A network of H servers, each providing a packet scale rate guarantee to flow f, the source of which transmits packets at least at its reserved rate during, provides an end-to-end throughput guarantee to flow f of the form:*

$$W_{f,H}(t_1, t_2) > r_f(t_2 - t_1) - r_f \left[\sum_{j=1}^{H} \epsilon_{f,j} + \sum_{j=1}^{H-1} \pi_j + (H+1)\frac{l^{max}_f}{r_f} \right]$$

where r_f is the rate reserved for flow f and $\epsilon_{f,j}$ is the latency term in the packet scale rate guarantee of server j.

A network of core-stateful fair servers, therefore, provides an end-to-end throughput guarantee characterized by: $\gamma^{net}_{f,H} = \sum_{j=1}^{H} \epsilon_{f,j} + \sum_{j=1}^{H-1} \pi_j + (H+1)\frac{l^{max}_f}{r_f}$.

[1]The analysis in [5] is conducted for networks with zero link propagation latencies—it can, however, be extended to networks with non-zero propagation latencies.

3.3 End-to-end Fairness Guarantees

3.3.1 Background

Single-server Fairness Guarantees

In any time interval during which flows are backlogged[2], an ideal fair server provides throughput to flows *exactly* in proportion to their reserved rates. This idealized notion of fairness, however, is infeasible to realize in a packetized system. Fair algorithms for packet scheduling, instead, guarantee an upper bound on the difference in the normalized throughput (weighted by the reserved rate) received by flows at a server in intervals during which they are continuously backlogged [22]. This notion of a *fairness guarantee* is formally defined as follows:

Definition 5 *The scheduling algorithm at node j is said to provide a fairness guarantee if in any time interval $[t_1, t_2]$ during which two flows f and m are continuously backlogged, the number of bits of flows f and m transmitted by the server, $W_{f,j}(t_1, t_2)$ and $W_{m,j}(t_1, t_2)$ respectively, satisfy:*

$$\left| \frac{W_{f,j}(t_1, t_2)}{r_f} - \frac{W_{m,j}(t_1, t_2)}{r_m} \right| \leq U_{j,\{f,m\}} \tag{3.5}$$

where r_f and r_m are the rates reserved for flows f and m respectively, and $U_{j,\{f,m\}}$ is the unfairness measure—a constant that depends on the scheduling algorithm and traffic characteristics at server j.

Different fair scheduling algorithms [6, 7, 22, 26, 47] differ in the value of $U_{j,\{f,m\}}$, the unfairness measure. Table 3.2 lists the $U_{j,\{f,m\}}$ values for several known fair scheduling algorithms.

Fair scheduling algorithms, in addition to providing throughput guarantees to backlogged flows at short time-scales, allocate *idle* link capacity to competing

[2]Fairness in throughput allocation is usually defined only with respect to flows that are backlogged. This is because the throughput of a non-backlogged flow may be constrained by the source traffic rather than by the allocation of link capacity.

flows in proportion to their weights (or reserved rates). This is desirable from an economic perspective. Consider, for instance, the case when a network provider charges its customers based on their reserved bandwidth. In such a network, if a user A pays twice as much as user B, then A expects the network to allocate bandwidth in the ratio 2:1 to users A and B; any other allocation would be considered unfair. Fair scheduling algorithms allow a network to ensure this proportionate allocation property independent of the amount of available bandwidth.

End-to-end Fairness Guarantees

Note that from an end-user perspective, it is more important to define a notion of *end-to-end fairness.* Additionally, from a network provider's perspective, quantification of end-to-end fairness guarantees is necessary for offering service level agreements (SLAs) to customers. In Definition 6, we generalize Definition 5 to an end-to-end fairness guarantee. Observe that the fairness property is meaningful only across flows that share a resource; thus, the notion of end-to-end fairness is defined only across flows that share the same end-to-end network path.

Definition 6 *A network is said to provide an end-to-end fairness guarantee if in any time interval $[t_1, t_2]$ during which two flows, f and m, that traverse the same network path of length H hops, are continuously backlogged at the* **<u>first</u>** *server, the number of bits of flows f and m that depart the network, $W_{f,H}(t_1, t_2)$ and $W_{m,H}(t_1, t_2)$ respectively, satisfy:*

$$\left| \frac{W_{f,H}(t_1, t_2)}{r_f} - \frac{W_{m,H}(t_1, t_2)}{r_m} \right| \leq U^{net}_{H,\{f,m\}} \tag{3.6}$$

where $U^{net}_{H,\{f,m\}}$ is a constant that depends on the server and traffic characteristics at the different hops in the end-to-end network path.

There is no analysis in the literature that derives an upper bound on the unfairness measure, $U^{net}_{H,\{f,m\}}$, for a network of fair servers. It may seem tempting to reason that, since the throughput of a flow is determined by the rate allocated at a "bottleneck" server along its path, the end-to-end fairness guarantee for two flows would be the same as the fairness guarantee provided by the bottleneck server. Unfortunately, such a reasoning is incorrect. This is because, end-to-end throughput received by a flow at short time-scales depends on the queuing delay suffered by its packets. At short time-scales, individual packets of flows may encounter queuing delay at *several* servers, even when at large time-scales, a single bottleneck server may determine the average bandwidth allocated to a flow. Therefore, the end-to-end fairness guarantee is not the same as the fairness guarantee provided by any single server.

Our Contributions We present the *first* end-to-end fairness analysis of a network of routers, each employing a fair scheduling algorithm. We argue that it is difficult to extend existing single-node fairness analysis to an end-to-end analysis of a network where each node may employ a different fair scheduling algorithm. We then present a two-step approach for end-to-end fairness analysis of heterogeneous networks. First, we define a class of scheduling algorithms, referred to as the *Fair Throughput (FT)* class, and prove that most known fair scheduling algorithms belong to this class. Second, we develop an analysis methodology for deriving the end-to-end fairness bounds for a network of FT servers. Our analysis is general and can be applied to heterogeneous networks where different nodes employ different scheduling algorithms from the FT class.

3.3.2 End-to-end Fairness Analysis: Challenges

As mentioned earlier, designers of individual fair scheduling algorithms generally prove fairness bounds ($U_{j,\{f,m\}}$) with respect to throughput achieved by different flows only at a *single* server [6, 22, 26]; the literature does not contain analysis that prove fairness bounds ($U^{net}_{H,\{f,m\}}$) for the *end-to-end* throughput achieved by flows in a network of fair servers. Unfortunately, extending existing single-server fairness analyses of fair scheduling algorithms to an end-to-end analysis is not straightforward. This is due to two main reasons:

1. The literature contains single-server fairness analyses only for *specific* scheduling algorithms [6, 7, 22, 26]. In a wide-area network, however, each router may employ a different scheduling algorithm. Hence, an end-to-end fairness analysis should be applicable to such heterogeneous networks.

2. Most single-server analyses of fair scheduling algorithms presented in the literature derive fairness bounds only over intervals during which the concerned flows are simultaneously and continuously backlogged. Due to the variability in the delay experienced by packets at a server, traffic gets distorted as it traverses through the network. Therefore, flows may not be simultaneously or continuously backlogged at all the servers along the path and during all time intervals, even if they are at the first server. Hence, it is not straightforward to apply existing single-server analyses to determine bounds on end-to-end network fairness.

We address these challenges in two steps. First, we define a general class of *Fair Throughput (FT)* servers; we show that most of the known fair scheduling algorithms belong to this class (Section 3.3.3). Second, we develop an analysis methodology for deriving end-to-end fairness bounds for a network of FT servers (Section 3.3.4). Our analysis is general, and can be applied to networks where different nodes employ different scheduling algorithms from the FT class.

3.3.3 The Class of Fair Throughput Servers

Recall that the fairness guarantee in Definition 5 is applicable only to time intervals during which both flows are continuously backlogged. As discussed in Section 3.3.2, there may be time intervals during which one or both of the flows may not be continuously backlogged at subsequent servers. To facilitate fairness analysis during such time intervals at subsequent servers, we define the class of *Fair Throughput (FT)* scheduling algorithms [36]. An algorithm in the FT class provides a stronger notion of the per-node fairness guarantee—if a flow m is continuously backlogged during an interval, then the normalized throughput received by *any* other flow f (whether continuously backlogged or not during the interval) does not exceed the normalized throughput received by flow m by more than a bounded quantity. We formalize this notion below.

Definition 7 *The scheduling algorithm at node j belongs to the class of Fair Throughput servers if in any time interval $[t_1, t_2]$ during which a flow m is continuously backlogged, the number of bits transmitted by the server for any flow f and flow m, $W_{f,j}(t_1, t_2)$ and $W_{m,j}(t_1, t_2)$ respectively, satisfy:*

$$\frac{W_{f,j}(t_1, t_2)}{r_f} \leq \frac{W_{m,j}(t_1, t_2)}{r_m} + I_{j,m,f}$$

where $I_{j,m,f}$ is a constant that depends on the server and traffic characteristics at node j.

From Definition 5 and Definition 7, it is easy to observe that all FT algorithms also provide fairness guarantees with $U_{j,\{f,m\}} = \max(I_{j,m,f}, I_{j,f,m})$.

The definition of the class of FT scheduling algorithms is fairly general, and most well-known fair scheduling algorithms—such as Generalized Processor Sharing (GPS) [47], Self-clocked Fair Queuing (SCFQ) [22], Start-time Fair Queuing (SFQ) [26], Worst-case Weighted Fair Queuing (WF^{2}Q) [6]— belong to this class. We derive the $U_{j,\{f,m\}}$ and $I_{j,m,f}$ values for these algorithms below, and summarize them in Table 3.2. GPS, by definition [47], provides ideal fairness with $U_{j,\{f,m\}} = I_{j,m,f} = 0$.

FT algorithm	$U_{j,\{f,m\}}$	$I_{j,m,f}$
GPS	0	0
SCFQ	$\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}$	$\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}$
SFQ	$\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}$	$\frac{l_f^{max}}{r_f} + \frac{l_m^{max}}{r_m}$
WF^{2}Q	$l^{max}\left(\frac{1}{r_f} + \frac{1}{r_m} - \frac{1}{C}\right)$	$\frac{l^{max}}{r_m} + l_f^{max}\left(\frac{1}{r_f} - \frac{1}{C}\right)$

Table 3.2: Unfairness measures for some FT algorithms

SCFQ [22]

Let $v(t_1, t_2)$ be the difference in system virtual times at t_1 and $t_2 \geq t_1$. Let $v_f(t_1, t_2)$ be the difference in virtual time of flow f at t_1 and t_2.

Since flow m is continuously backlogged throughout $[t_1, t_2]$, we have from Corollary 4 of [22]:

$$\frac{W_m(t_1, t_2)}{r_m} \geq v(t_1, t_2) - \frac{l_m^{max}}{r_m} \tag{3.7}$$

Consider flow f. Break $[t_1, t_2]$ into sub-intervals belonging to two sets: B, the set of sub-intervals during which f is continuously backlogged, and NB, the set of sub-intervals during which f is *not* backlogged. From Definition 5, the differential service lag function for flow f, δ_f, is defined as:

$$\begin{aligned}\delta_f(t_1, t_2) &= v(t_1, t_2) - v_f(t_1, t_2) \\ &= v(t_1, t_2) - \sum_{[t', t"] \in B} v_f(t', t") - \sum_{[t', t"] \in NB} v_f(t', t") \\ &= v(t_1, t_2) - \sum_{[t', t"] \in B} W_f(t', t") - \sum_{[t', t"] \in NB} v(t', t")\end{aligned}$$

where we use Definition 3 and 4 of [22]. From Lemma 2 of [22], we know that $v(t)$ is a non-decreasing function of time. Therefore,

$$\begin{aligned}\delta_f(t_1, t_2) &\leq v(t_1, t_2) - \sum_{[t', t"] \in B} W_f(t', t") \\ &\leq v(t_1, t_2) - W_f(t_1, t_2)\end{aligned} \tag{3.8}$$

From Theorem 1 of [22], we know that $\delta_f(t_1,t_2) \geq \frac{l_f^{max}}{r_f}$. Therefore, we get:

$$v(t_1,t_2) \quad \geq \quad W_f(t_1,t_2) - \frac{l_f^{max}}{r_f} \tag{3.9}$$

From (3.7) and (3.9) we get:

$$\frac{W_f(t_1,t_2)}{r_f} \quad \leq \quad \frac{W_m(t_1,t_2)}{r_m} + \frac{l_m^{max}}{r_m} + \frac{l_f^{max}}{r_f}$$

Therefore, for an SCFQ server, $U_{j,\{f,m\}} = I_{j,m,f} = \frac{l_m^{max}}{r_m} + \frac{l_f^{max}}{r_f}$.

SFQ [26]

Let v_1 and v_2, respectively, be the *virtual times* at t_1 and t_2. Since flow m is backlogged throughout $[t_1,t_2]$, we have from Lemma 1 of [26]:

$$W_m(t_1,t_2) \quad \geq \quad r_m(v_2 - v_1) - l_m^{max} \tag{3.10}$$

From Lemma 2 of [26], we have for flow f:

$$W_f(t_1,t_2) \quad \leq \quad r_f(v_2 - v_1) + l_f^{max} \tag{3.11}$$

From (3.10) and(3.11), we get:

$$\begin{aligned} \frac{W_m(t_1,t_2)}{r_m} + \frac{l_m^{max}}{r_m} \geq v_2 - v_1 &\geq \frac{W_f(t_1,t_2)}{r_f} - \frac{l_f^{max}}{r_f} \\ \Rightarrow \qquad \frac{W_f(t_1,t_2)}{r_f} &\leq \frac{W_m(t_1,t_2)}{r_m} + \frac{l_m^{max}}{r_m} + \frac{l_f^{max}}{r_f} \end{aligned}$$

Therefore, for an SFQ server, $U_{j,\{f,m\}} = I_{j,m,f} = \frac{l_m^{max}}{r_m} + \frac{l_f^{max}}{r_f}$.

WF^{2}Q [6]

According to Theorem 1 in [6], the work done for a flow f at a WF^{2}Q server, W_f, is related in the following ways to the work done for the flow in a corresponding GPS [37] server, W_f^{GPS}:

$$W_f^{GPS}(t_1,t_2) - W_f^{WF^2Q}(t_1,t_2) \quad \leq \quad l^{max} \tag{3.12}$$

$$W_f^{WF^2Q}(t_1,t_2) - W_f^{GPS}(t_1,t_2) \quad \leq \quad (1 - \frac{r_f}{C})l_f^{max} \tag{3.13}$$

Given two flows f and m in a GPS server, of which flow m is continuously backlogged during a time interval $[t_1, t_2]$ (and flow f is not necessarily backlogged), we have the following:

$$\frac{W_f^{GPS}(t_1,t_2)}{r_f} \leq \frac{W_m^{GPS}(t_1,t_2)}{r_m}$$

Using (3.12) and (3.13), we get:

$$\begin{aligned}
\frac{W_f^{WF^2Q}(t_1,t_2)}{r_f} - (1-\frac{r_f}{C})\frac{l_f^{max}}{r_f} \leq \frac{W_f^{GPS}(t_1,t_2)}{r_f} &\leq \frac{W_m^{GPS}(t_1,t_2)}{r_m} \leq \frac{W_m^{WF^2Q}(t_1,t_2)}{r_m} + \frac{l^{max}}{r_m} \\
\Rightarrow \qquad \frac{W_f^{WF^2Q}(t_1,t_2)}{r_f} &\leq \frac{W_m^{WF^2Q}(t_1,t_2)}{r_m} + \frac{l^{max}}{r_m} + \frac{l_f^{max}}{r_f} - \frac{l_f^{max}}{C}
\end{aligned}$$

Therefore, for a WF^{2}Q server:

$$\begin{aligned}
I_{j,m,f} &= \frac{l^{max}}{r_m} + \frac{l_f^{max}}{r_f} - \frac{l_f^{max}}{C} \\
U_{j,\{f,m\}} &= \max\left\{\frac{l^{max}}{r_f} + l_m^{max}\left(\frac{1}{r_m} - \frac{1}{C}\right),\ \frac{l^{max}}{r_m} + l_f^{max}\left(\frac{1}{r_f} - \frac{1}{C}\right)\right\} \\
&\leq l^{max}\left(\frac{1}{r_m} + \frac{1}{r_f} - \frac{1}{C}\right)
\end{aligned}$$

We expect that any fair scheduling algorithm that provides per-node fairness guarantee (Definition 5) can be shown to belong to the FT class. This is because, during sub-intervals in which f is also backlogged, the difference in normalized throughput received by flows f and m is bounded (due to the fairness guarantee provided by the fair scheduling algorithm). On the other hand, during sub-intervals in which f is *not* backlogged, its throughput is zero, which can not exceed the throughput received by flow m. A formal proof for this assertion, however, is beyond the scope of this analysis.

3.3.4 End-to-end Analysis of FT Networks

Given a network of FT servers, our objective is to derive an upper-bound on $\left|\frac{W_{f,H}(t_1,t_2)}{r_f} - \frac{W_{m,H}(t_1,t_2)}{r_m}\right|$, the difference in normalized throughput received during any time interval $[t_1, t_2]$ by flows f and m that traverse the same end-to-end path of H servers, assuming that the flows are continuously-backlogged at the **first** server. In the following, we first present the methodology for conducting the end-to-end fairness analysis, and then present the formal lemmas, theorem, and their proofs. For simplicity of exposition, we assume zero propagation latencies on the links connecting servers while discussing the methodology; the formal analysis in the Lemmas and Theorem is, however, presented for the general case of links with non-zero propagation latencies.

Analysis Methodology

As mentioned in Section 3.3.2, one of the main challenges in extending single-server fairness analysis to an end-to-end analysis is that flows may not remain continuously or even simultaneously backlogged at subsequent servers. The question we would like to answer is: *given that the first server provides a fairness guarantee to the two backlogged flows, can we say something similar about the throughput received at the second, third, fourth, and so on, servers?* If we can relate the fairness guarantee provided at a server to the fairness guarantee provided at the *previous* server, then by using a recursive argument, we would be able to answer the above question in the affirmative. In particular, we need to answer the transformed question: *given* $U^{net}_{j,\{f,m\}}$*, the bound on difference in normalized throughput achieved at* j^{th} *server, can we compute* $U^{net}_{j+1,\{f,m\}}$*, the bound on difference in normalized throughput at the* $(j+1)^{th}$ *server?*

For any time interval $[t_1, t_2]$ at the $(j+1)^{th}$ server, we consider the following two cases:

- If *none* of the flows are backlogged at the *time instants* t_1 and t_2, then the throughput received by the two flows in the interval $[t_1, t_2]$ is the *same* as the throughput they receive at the previous server in this time interval (assuming zero propagation latencies). This is because, no packets that were received before t_1 get served in $[t_1, t_2]$ (since there is no backlog at t_1), and no packets received during $[t_1, t_2]$ get served after t_2 (no backlog at t_2 either).

Therefore, for such time intervals, the difference in normalized throughput of the two flows at server $j + 1$ has the same upper bound as that for server j.

- If either one of the flows is backlogged at server $j + 1$ at t_1 or t_2, then the number of packets of that flow served during $[t_1, t_2]$ at server $j + 1$ may be different from those served at server j. This is because, some packets that are backlogged at t_1 may get served in $[t_1, t_2]$, and some packets that arrive from server j in $[t_1, t_2]$ may not get served and may remain backlogged at t_2. The throughput of the flow during $[t_1, t_2]$ at server $j + 1$ is therefore determined not only by its throughput at server j, but also the backlogs at server $j + 1$ at times t_1 and t_2. It follows that to derive the fairness guarantee of server $j + 1$ during such time intervals, we additionally need to bound the difference in the normalized backlogs of the two flows, at both t_1 and t_2.

  Let us consider a time instant t_0, smaller than t_1, at which none of the flows are backlogged at server $j + 1$. The backlog of either flow at t_1 (or t_2) can be computed as the number of packets that arrive in $[t_0, t_1]$ (or $[t_0, t_2]$) minus the number of packets that are served in the same time interval. From the fairness guarantee of server j, we know that the difference in (normalized) number of packets that arrive at server $j + 1$ in $[t_0, t_1]$ (or $[t_0, t_2]$) for the two flows is bounded. It follows that to bound the difference in normalized backlogs at t_1 (or t_2) at server $j + 1$, we need to bound the difference in normalized throughput during $[t_0, t_1]$ (or $[t_0, t_2]$).

  To compute the difference in normalized throughput of flows f and m at server $j + 1$ during $[t_0, t_1]$ (or $[t_0, t_2]$), we consider the following two scenarios:

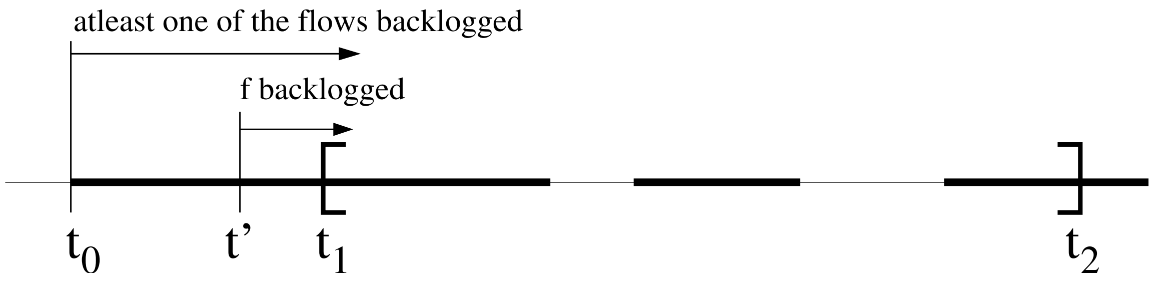


Figure 3.1: Reference for Lemma 2

– *Both flows are backlogged at t_1 (or t_2).*

Lemma 2 establishes a lower bound on the normalized throughput during $[t_0, t_1]$ of either flow (say, f) in terms of the normalized throughput of the other (flow m). The proof methodology for this lemma is based on the following two observations. Let $t' \in [t_0, t_1]$ be the *last* time instant before which f is non-backlogged (see Figure 3.1).

1. The throughput of flow f in $[t_0, t']$ at server $j+1$ is the same as its throughput in the same time interval at server j (since no backlogs at either t_0 or t'). At server j, the throughput of flow f is lower bounded in terms of the throughput of flow m (due to the fairness guarantee of server j). Further, the throughput of flow m at server $j+1$ during $[t_0, t']$ (no backlog at t_0) cannot exceed the corresponding throughput at server j.
2. Flow f is continuously backlogged in $[t', t_1]$ and Definition 7 can be applied to this interval to establish a lower bound on throughput of flow f in terms of throughput of flow m.

– *Only one of the flows is backlogged at t_1 (or t_2).*

As described above, Lemma 2 can derive a lower bound on the normalized throughput during $[t_0, t_1]$ of the *backlogged* flow (say, f) in terms of the throughput of the other (flow m).

To compute the reverse relation, that is, a lower bound on throughput of flow m, first observe that during $[t_0, t_1]$, the throughput of flow m at server $j+1$ is the same as its throughput at server j (since no backlogs at either t_0 or t_1). At server j, the throughput of flow m is lower bounded in terms of the throughput of flow f (due to the fairness guarantee of server j). Further, the throughput of flow f at server $j+1$ during $[t_0, t_1]$ (no backlog at t_0) cannot exceed its throughput at server j. These observations are used in Lemma 3 to compute a lower bound on the throughput of flow m in terms of that of flow f.

Therefore, by using Lemma 2 and Lemma 3, we can compute upper bounds on the difference in normalized throughput of the two flows during both $[t_0, t_1]$ and $[t_0, t_2]$. The sum of these two bounds then gives a candidate value of $U^{net}_{j+1,\{f,m\}}$, the upper bound on the difference in normalized throughput during $[t_1, t_2]$. Lemma 4 helps tighten this value, based on the fairness guarantee of the FT algorithm at server j.

Formal Analysis and Proofs

Using the methodology described above, we derive the end-to-end fairness guarantee of a network of FT servers in Theorem 3. The following lemmas, which are used in the proof analysis, are proved in the appendices. For the remainder of the analysis, we use $\pi_j(t)$ to denote the propagation latency experienced—on the link connecting server j and $j+1$—by the *last* packet of either flow received at server $j+1$ by time t.

Lemma 2 *If flow f is backlogged at server $j+1$ at time t and if t_0 is a time instant smaller than t such that neither f nor m is backlogged at t_0^-, and at least one is* *backlogged at t_0, then:*

$$\frac{W_{f,j+1}(t_0,t)}{r_f} \geq \frac{W_{m,j+1}(t_0,t)}{r_m} + a' - I_{j+1,f,m}$$

where $a' = W_{f,j}(t_0 - \pi_j(t_0), t' - \pi_j(t'))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t' - \pi_j(t'))/r_m$, and $t' \in [t_0, t]$ is the latest time instant at which f becomes backlogged.

Proof: Since packets arrive at server $j+1$ in the same order they are transmitted at server j, it follows that the packets received at server $j+1$ during $[t_0, t']$ are the packets transmitted from server j during $[t_0 - \pi_j(t_0), t' - \pi_j(t')]$.

Since f is not backlogged at t'^- and t_0^-, we have: $W_{f,j+1}(t_0, t') = W_{f,j}(t_0 - \pi_j(t_0), t' - \pi_j(t'))$. Further, since flow m is not backlogged at t_0^-, we have: $W_{m,j+1}(t_0, t') \leq W_{m,j}(t_0 - \pi_j(t_0), t' - \pi_j(t'))$.

Since $a' = W_{f,j}(t_0 - \pi_j(t_0), t' - \pi_j(t'))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t' - \pi_j(t'))/r_m$. Then, we have:

$$\frac{W_{f,j+1}(t_0,t')}{r_f} \geq \frac{W_{m,j+1}(t_0,t')}{r_m} + a' \tag{3.14}$$

Since flow f is continuously backlogged during $[t', t]$, and server $j+1$ belongs to the FT class, we have from Definition 7:

$$\frac{W_{m,j+1}(t',t)}{r_m} \leq \frac{W_{f,j+1}(t',t)}{r_f} + I_{j+1,f,m} \tag{3.15}$$

From (3.14) and (3.15), we get:

$$\frac{W_{f,j+1}(t_0,t)}{r_f} \geq \frac{W_{m,j+1}(t_0,t)}{r_m} + a' - I_{j+1,f,m}$$

■

Lemma 3 *If flow m is* **not** *backlogged at server $j+1$ at time t and if t_0 is a time instant smaller than t such that neither f nor m is backlogged at t_0^-, and at least one is backlogged at t_0, then:*

$$\frac{W_{m,j+1}(t_0,t)}{r_m} \geq \frac{W_{f,j+1}(t_0,t)}{r_f} - a$$

where $a = W_{f,j}(t_0 - \pi_j(t_0), t - \pi_j(t))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t - \pi_j(t))/r_m$.

Proof: Since packets arrive at server $j+1$ in the same order they are transmitted at server j, it follows that the packets received at server $j+1$ during $[t_0, t]$ are the packets transmitted from server j during $[t_0 - \pi_0, t - \pi]$.

Since m is not backlogged at t and t_0^-, we have: $W_{m,j+1}(t_0, t) = W_{m,j}(t_0 - \pi_0, t - \pi)$. Further, since f is not backlogged at t_0^-, we have: $W_{f,j+1}(t_0, t) \leq W_{f,j}(t_0 - \pi_0, t - \pi)$.

Since $a = W_{f,j}(t_0 - \pi_0, t - \pi)/r_f - W_{m,j}(t_0 - \pi_0, t - \pi)/r_m$, we have:

$$\frac{W_{m,j+1}(t_0, t)}{r_f} \geq \frac{W_{f,j+1}(t_0, t)}{r_m} - a$$

■

Lemma 4 *If during a time interval $[t_0, t]$, the numbers of bits transmitted by a server for flow f and m satisfy (5) with unfairness measure H', and if $a_1 = W_{f,j}(t_0, t_1)/r_f - W_{m,j}(t_0, t_1)/r_m$, and $a_2 = W_{f,j}(t_0, t_2)/r_f - W_{m,j}(t_0, t_2)/r_m$, where $t_0 \leq t_1 \leq t_2 \leq t$, then*

$$-H' \leq a_2 - a_1 \leq H'$$

Proof:

$$\begin{aligned}
\frac{W_{f,j}(t_1, t_2)}{r_f} &= \frac{W_{f,j}(t_0, t_2)}{r_f} - \frac{W_{f,j}(t_0, t_1)}{r_f} \\
&= \frac{W_{m,j}(t_0, t_2)}{r_m} + a_1 - \frac{W_{m,j}(t_0, t_1)}{r_m} - a_2 \\
&= \frac{W_{m,j}(t_1, t_2)}{r_m} + a_1 - a_2
\end{aligned}$$

Since the number of bits transmitted during the interval $[t_1, t_2]$ should satisfy (5) with unfairness measure H', it follows that: $-H' \leq a_2 - a_1 \leq H'$. ■

Theorem 3 *If the throughput obtained by two flows, f and m, at the first node in a network of FT servers satisfies (5), and if they share the same end-to-end path* *of H hops, then during any time interval $[t_1, t_2]$, the end-to-end throughput of the flows are related as:*

$$\left| \frac{W_{f,H}(t_1,t_2)}{r_f} - \frac{W_{m,H}(t_1,t_2)}{r_m} \right| \leq U_{1,\{f,m\}} + \sum_{h=2}^{H} (I_{h,f,m} + I_{h,m,f}) \quad (3.16)$$

Proof: The proof is by induction on j.

Base Case: $j = 1$. Since the flows f and m are assumed to be simultaneously and continuously backlogged at the first server, and since the first server belongs to the FT class, we have from (5):

$$\left| \frac{W_{f,1}(t_1,t_2)}{r_f} - \frac{W_{m,1}(t_1,t_2)}{r_m} \right| \leq U_{1,\{f,m\}} \quad (3.17)$$

Therefore, (3.16) is true for the base case.

Induction Hypothesis: Assume (3.16) is true for all servers $1, \ldots, j$.

Induction Step: We need to show that (3.16) holds at server $j+1$. Without loss of generality, assume that $\frac{W_{f,j+1}(t_1,t_2)}{r_f} \geq \frac{W_{m,j+1}(t_1,t_2)}{r_m}$. It follows that to prove (3.16), we only need to show that:

$$\frac{W_{f,1}(t_1,t_2)}{r_f} \leq \frac{W_{m,1}(t_1,t_2)}{r_m} + U_{1,\{f,m\}} + \sum_{h=2}^{j+1} (I_{h,f,m} + I_{h,m,f}) \quad (3.18)$$

Let $t_0 \leq t_1$ be the *largest* time instant such that none of the two flows are backlogged at server $j+1$ at t_0^- (see Figure 3.2).

<u>Difference in normalized throughput during $[t_0, t_1]$:</u>

Consider the following two cases at t_1:

- If flow f is backlogged at t_1 (see Figure 3.2), then from Lemma 2, there exists $t_3 \in [t_0, t_1]$, such that:

$$\frac{W_{f,j+1}(t_0,t_1)}{r_f} \geq \frac{W_{m,j+1}(t_0,t_1)}{r_m} + a_3 - I_{j+1,f,m}$$

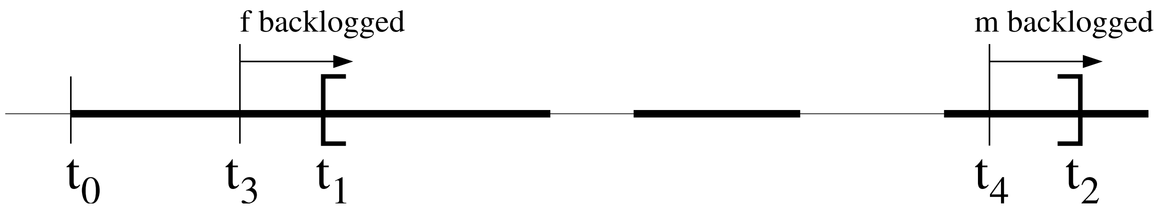


Figure 3.2: Reference for proof of Theorem 3

where $a_3 = W_{f,j}(t_0 - \pi_j(t_0), t_3 - \pi_j(t_3))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t_3 - \pi_j(t_3))/r_m$.

- If flow f is *not* backlogged at t_1, then from Lemma 3, we have:

$$\begin{aligned}\frac{W_{f,j+1}(t_0,t_1)}{r_f} &\geq \frac{W_{m,j+1}(t_0,t_1)}{r_m} + a_1 \\ &\geq \frac{W_{m,j+1}(t_0,t_1)}{r_m} + a_1 - I_{j+1,f,m}\end{aligned}$$

where $a_1 = W_{f,j}(t_0 - \pi_j(t_0), t_1 - \pi_j(t_1))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t_1 - \pi_j(t_1))/r_m$.

Therefore, in either case, there exists some $t' \in [t_0, t_1]$, such that:

$$\frac{W_{f,j+1}(t_0,t_1)}{r_f} \geq \frac{W_{m,j+1}(t_0,t_1)}{r_m} + a' - I_{j+1,f,m} \tag{3.19}$$

where $a' = W_{f,j}(t_0 - \pi_j(t_0), t' - \pi_j(t'))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t' - \pi_j(t'))/r_m$.

Difference in normalized throughput during $[t_0, t_2]$:

Consider the following two cases at t_2:

- If flow m is backlogged at t_2 (see Figure 3.2), then from Lemma 2, there exists $t_4 \in [t_0, t_2]$, such that:

$$\frac{W_{m,j+1}(t_0,t_2)}{r_m} \geq \frac{W_{f,j+1}(t_0,t_2)}{r_f} - a_4 - I_{j+1,m,f}$$

where $a_4 = W_{f,j}(t_0 - \pi_j(t_0), t_4 - \pi_j(t_4))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t_4 - \pi_j(t_4))/r_m$.

- If flow m is *not* backlogged at t_2, then from Lemma 3, we have:

$$\begin{aligned} \frac{W_{m,j+1}(t_0,t_2)}{r_m} &\geq \frac{W_{f,j+1}(t_0,t_2)}{r_f} - a_2 \\ &\geq \frac{W_{f,j+1}(t_0,t_2)}{r_f} - a_2 - I_{j+1,m,f} \end{aligned}$$

where $a_2 = W_{f,j}(t_0 - \pi_j(t_0), t_2 - \pi_j(t_2))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t_2 - \pi_j(t_2))/r_m$.

Therefore, in either case, there exists some $t" \in [t_0, t_2]$, such that:

$$\frac{W_{m,j+1}(t_0,t_2)}{r_m} \geq \frac{W_{f,j+1}(t_0,t_2)}{r_f} - a" - I_{j+1,m,f} \qquad (3.20)$$

where $a" = W_{f,j}(t_0 - \pi_j(t_0), t" - \pi_j(t"))/r_f - W_{m,j}(t_0 - \pi_j(t_0), t" - \pi_j(t"))/r_m$.

Difference in normalized throughput during $[t_1, t_2]$:

From (3.19) and (3.20), we get:

$$\begin{aligned} \frac{W_{f,j+1}(t_1,t_2)}{r_f} &= \frac{W_{f,j+1}(t_0,t_2)}{r_f} - \frac{W_{f,j+1}(t_0,t_1)}{r_f} \\ &\leq \frac{W_{m,j+1}(t_0,t_2)}{r_m} + a" + I_{j+1,m,f} - \frac{W_{m,j+1}(t_0,t_1)}{r_m} - a' + I_{j+1,f,m} \end{aligned} \qquad (3.21)$$

From the induction hypothesis and Lemma 4, we know that: $-(U_{1,\{f,m\}} + \sum_{h=2}^{j}(I_{h,f,m} + I_{h,m,f})) \leq a" - a' \leq U_{1,\{f,m\}} + \sum_{h=2}^{j}(I_{h,f,m} + I_{h,m,f})$. Therefore, from (3.21), it can be seen that (3.18) holds at server $j+1$.

Hence, the theorem follows by mathematical induction. ■

Theorem 3 states that the unfairness measure for a network of servers is a linear function of the unfairness measures of the individual servers. An interesting property of the guarantee is that unlike end-to-end guarantees on delay (Theorem 1) and throughput (Theorem 2), which in addition, are linear functions of link propagation latencies, the bound on end-to-end fairness in Theorem 3 is independent of link propagation latencies. The end-to-end fairness guarantees of wide-area networks may therefore be similar to those of local-area networks, as long as the number of servers on a typical path are similar.

It is important to observe that the analysis of Theorem 3 can be used to derive end-to-end fairness guarantees even for flows that are *not* continuously backlogged at the first servers. The only condition that needs to be satisfied at the first server is that the difference in normalized throughput of flow f and m at the first server is bounded. This may be true even if the flows are not continuously backlogged, for instance in the case when the difference in (normalized) number of bits that arrive from the respective sources is bounded. To obtain the end-to-end fairness guarantee for such cases, $U_{1,\{f,m\}}$ is replaced in (3.16) by the bound on difference in normalized throughput at the first server.

Note 1 *Note that we have assumed that r_f, the rate reserved for flow f, is the same at all routers on its path. There are many ways to compute r_f. For instance, r_f could be the minimum rate acceptable to the end-user, or it could be the rate determined according to the* max-min *fair rate allocation scheme—such a scheme allocates to a flow a rate at each router that is no more than the fair rate allocated to that flow at its bottleneck router. The problem of determining the rate r_f is orthogonal to the problem of end-to-end fairness analysis, and is outside the scope of this analysis.*

Our analysis of a network of FT servers, thus, helps us understand the end-to-end fairness guarantees that can be provided using a core-stateful network architecture.

3.4 Summary

The single-server delay, throughput, and fairness guarantees provided by several sophisticated scheduling algorithms are well-understood. In this chapter, we focus on understanding the end-to-end guarantees that can be provided using a core-stateful network architecture.

We build on past work to understand end-to-end delay and throughput guarantees, and present a novel analysis to derive end-to-end fairness guarantees of core-stateful networks. Our analysis results in the *first* tight bound on end-to-end fairness for any network of fair servers. We first argue that it is difficult to extend existing single-node fairness analysis to an end-to-end analysis of a network where each node may employ a different fair scheduling algorithm. We then present a two-step approach for end-to-end fairness analysis of heterogeneous networks. First, we define a class of scheduling algorithms, referred to as the *Fair Throughput (FT)* class, and prove that most known scheduling algorithms belong to this class. Second, we develop an analysis methodology for deriving the end-to-end fairness bounds for a network of FT servers. Our analysis is general and can be applied to heterogeneous networks where different nodes employ different scheduling algorithms from the FT class.

Having understood the end-to-end delay, throughput, and fairness guarantees that can be provided in core-stateful networks, we next explore—in Chapters 4-6—the design of core-stateless networks that provide similar guarantees.

Chapter 4

Core-stateless Guaranteed Rate Networks

Providing delay guarantees is fundamental to providing other types of service guarantees. In this chapter, we take our first step in designing core-stateless network architectures by designing one that provides end-to-end delay guarantees. In what follows, we first describe past work in this direction, and then present our methodology to derive the core-stateless version of any GR network [24] that provides exactly the same end-to-end delay guarantee.

4.1 Past Work

CJVC In [59], a *core-stateless* version, namely Core Jitter Virtual Clock (CJVC), of the Jitter Virtual Clock scheduling algorithm has been proposed. CJVC provides the same delay guarantees as Jitter Virtual Clock, while maintaining and using per-flow state only at the edges of the network. CJVC, just like JVC, is non-work-conserving, which implies that it employs a *constant bit-rate* (CBR) per-flow shaper at every router. Queue lengths observed in a network of such servers are generally smaller than in networks of work-conserving schedulers. This further reduces the computation complexity of implementing such a scheduler.

Unfortunately, the non-work-conserving nature of the CJVC algorithm limits the extent of *statistical multiplexing* gains that the framework can benefit from. This is because non-work-conserving algorithms shape the traffic to the maximum of the reserved rate and sending rate for that flow; when a flow sends a burst of packets at a rate greater than its reserved rate, extra packets are held until their eligibility time, even if idle bandwidth is available for transmitting these packets. Such an approach may under-utilize available network resources. As argued in Chapter 1, to meet rising traffic demands, it is important for networks to utilize their resources efficiently. Hence, core-stateless networks that provide delay guarantees and also are work-conserving are desirable.

In order to design such networks, we first ask the question: *can the techniques used in deriving CJVC from JVC be applied directly to derive core-stateless versions of work-conserving algorithms like Virtual Clock?* In Section 4.1.1, we show that this is not the case.

4.1.1 Extending the CJVC Design Methodology to Virtual Clock

We begin by first outlining the technique of deriving CJVC from JVC. The Jitter Virtual-Clock (JVC) algorithm assigns to packet p_f^k a *deadline* $D_{f,j}^k$ at each server j as follows:

$$\begin{aligned} e_{f,j}^1 &= a_{f,j}^1 && (4.1)\\ e_{f,j}^k &= \max\left(a_{f,j}^k + g_{f,j-1}^k, D_{f,j}^{k-1}\right) && (4.2)\\ D_{f,j}^k &= e_{f,j}^k + \frac{l_f^k}{r_f}, \qquad j,k \geq 1 && (4.3) \end{aligned}$$

where $g_{f,j-1}^k$ is the amount of time by which the packet is transmitted before its deadline at server $j-1$, and $e_{f,j}^k$ denotes the *eligibility time*—the time at which packet

p_f^kbecomes eligible for transmission—at server j. The JVC algorithm transmits packets in the increasing order of their deadlines.

As is evident, to derive the eligibility time and hence the deadline of a packet, JVC—and many other *guaranteed rate* algorithms such as Virtual Clock—requires the server to remember for each flow the deadline $D_{f,j}^{k-1}$ of the previous packet (used in the max-term computation of Equation (4.2)). In deriving the Core-stateless Jitter Virtual Clock (CJVC) algorithm, the authors of [59] observe that the need to maintain such per-flow state at *core* routers can be eliminated by introducing a per-packet *slack variable,* δ_f^k, that accounts for the maximum difference between the two quantities in the max-term computation. For CJVC, this translates to deriving δ_f^k such that for servers $j \geq 2$:

$$\delta_f^k + a_{f,j}^k + g_{f,j-1}^k \quad \geq \quad D_{f,j}^{k-1} \tag{4.4}$$

In [59], authors derive a lower bound on the value of δ_f^k; further, they demonstrate that by using this lower bound, a network of CJVC server provide the same end-to-end delay guarantee as the network of JVC servers. The derivation of the lower bound on δ_f^k exploits a key property of the JVC algorithm: the non-work conserving JVC algorithm at each server shapes flows to their reserved rate and delays any packets arriving earlier until their *eligibility time.* This property enables the JVC algorithm to bound jitter, which in turn bounds the difference between the deadline of a packet and the eligibility time of the next packet of the same flow. Thus, $(D_{f,j}^{k-1} - a_{f,j}^k - g_{f,j-1}^k)$ is upper-bounded for all k — this is used as a lower bound for δ_f^k.

Now consider Virtual Clock [65]—a work-conserving packet scheduling algorithm. The Virtual Clock algorithm assigns to each packet p_f^k a *virtual clock* value $VC_{f,j}^k$ at server j as follows:

$$VC_{f,j}^1 \quad = \quad a_{f,j}^1 + \frac{l_f^1}{r_f^1} \tag{4.5}$$

$$VC^k_{f,j} = \max(a^k_{f,j}, VC^{k-1}_{f,j}) + \frac{l^k_f}{r_f} \tag{4.6}$$

Packets are transmitted in the increasing order of their *virtual clock* values.

To eliminate the need for maintaining $VC^{k-1}_{f,j}$ in routers, if we apply the technique used to derive CJVC from JVC, then we need to compute a per-packet *slack variable* δ^k_f such that for servers $j \geq 2$:

$$\delta^k_f \geq VC^{k-1}_{f,j} - a^k_{f,j} \tag{4.7}$$

Unfortunately, in a network of work-conserving Virtual Clock servers, unlike in a JVC network, packets can arrive back-to-back at a core router. The larger the number of back-to-back arrivals, the greater is the difference between the deadline of the penultimate packet of the burst and the arrival time of the last packet in the burst. Hence, to account for such bursty arrivals, the lower bound on δ^k_f grows with k. Consequently, we find that a network of core-stateless Virtual Clock servers (derived using the technique presented in [59]) does not preserve the delay guarantee of the corresponding network of VC servers. Appendix A presents a detailed analysis of this effect.

In what follows, we present a general methodology to derive the core-stateless version of any *Guaranteed Rate* (GR) [24] scheduling algorithm. Our methodology applies both to work-conserving and non-work-conserving algorithms in GR; further, we demonstrate that a network of such core-stateless GR servers provide the same delay guarantee as the corresponding network of GR servers.

We first present our methodology for deriving the core-stateless version of a specific GR algorithm, namely, Virtual Clock. Later, we generalize the methodology to the design of core-stateless versions of any GR network.

4.2 Design Methodology

Virtual Clock (VC) belongs to the GR class (Definition 1). Therefore, the end-to-end delay guarantee provided to packet p_f^k of a flow f by a network of VC servers is characterized in terms of the upper bound on $VC_{f,H}^k$, the deadline of p_f^k at the last server H. The deadline of packet p_f^k at any node, on the other hand, is computed using the deadline of the previous packet p_f^{k-1} at the same node (see Equation (4.6)). In order to compute deadlines for packets of flow f, therefore, a VC server needs to maintain per-flow state information about the deadline assigned to the most recent packet of that flow. In a core-stateless architecture, however, core routers do not maintain per-flow state, and hence, can not compute deadlines as given by Equation (4.6). So the problem of providing end-to-end delay guarantees in a core-stateless network, similar to a Virtual Clock network, boils down to answering the question: *how can one compute deadlines in a core-stateless network that does not maintain per-flow state?*

4.2.1 Key Insight

Observe that in order to compute deadlines without using per-flow state, we need to remove the dependence of the deadline of a packet on the state of the previous packet, and compute it using only the state of the *same* packet.

Our methodology for providing end-to-end delay guarantees without maintaining per-flow state at core routers is based on the following observations:

1. Although the deadline of a packet depends on the state of the previous packet at the same node, the *upper bound* on the deadline can be computed using only the state of the *same* packet at the *previous* node.

2. By using the above observation recursively, the upper bound on the deadline of a packet at *any* node can be computed using the state of the *same* packet

at the *first* node in the network.

3. Since the first node is an edge router of a core-stateless architecture, it does maintain per-flow state. This implies that the first node is capable of computing the upper bound on the deadline of a packet at *any* subsequent node in the network.

4. Finally, we show that if core nodes use the upper bound on deadlines, instead of the actual deadlines, to schedule packets for transmission, the network provides the same end-to-end delay guarantee.

Based on these insights, we formally present our methodology next.

4.3 Design of a CSVC Network

The following lemma (stated and proved in [24]) provides the key property that we use in the design of core-stateless versions of Virtual Clock and other GR algorithms:

Lemma 5 *If the scheduling algorithm at server j in a network belongs to the class of GR algorithms for flow f, then*

$$GRC^k_{f,j} \leq GRC^k_{f,j-1} + \max_{i \in [1..k]} \frac{l^i_f}{r^i_f} + \pi_{j-1} + \beta_{j-1}$$

where π_{j-1} is the propagation delay on the link connecting node $(j-1)$ and j.

Note that for Virtual Clock, $VC^k_{f,j} = GRC^k_{f,j}$. From the above lemma, therefore, we get:

$$VC^k_{f,j} \leq VC^k_{f,j-1} + \max_{i \in [1..k]} \frac{l^i_f}{r^i_f} + \pi_{j-1} + \frac{l^{max}_{j-1}}{C_{j-1}} \tag{4.8}$$

The upper bound on deadline of packet p^k_f at node j can, therefore, be computed using its deadline at node $j-1$.

4.3.1 Definition

We define the *core virtual clock* of p_f^k at server j, $VCore_{f,j}^k$, as follows:

$$VCore_{f,1}^k = VC_{f,1}^k \tag{4.9}$$

$$VCore_{f,j}^k = VCore_{f,j-1}^k + \beta_{f,j-1} + \pi_{j-1} + \max_{i\in[1..k]} \frac{l_f^i}{r_f^i}, \qquad j \geq 2 \tag{4.10}$$

where $\beta_{f,j-1} = \frac{l_{j-1}^{max}}{C_{j-1}}$. We define server j to be a *Core-Stateless Virtual Clock* server if it schedules packets in the increasing order of their $VCore_{f,j}^k$ values. From (4.8)-(4.10), it is easy to see that $VCore_{f,j}^k \geq VC_{f,j}^k$.

Thus, we have eliminated the need to maintain $VC_{f,j}^{k-1}$, to compute the max term of Equation (4.6). Additionally, by enabling edge routers to encode the rate r_f^k, and enabling server $j-1$ to encode $VCore_{f,j-1}^k$ in the packet, we can eliminate the need to maintain *any* per-flow state in the core routers of the network.

We now prove that the end-to-end delay bound of a network of such Core-Stateless Virtual Clock (CSVC) servers is the same as that of a network of Virtual Clock servers.

4.3.2 Delay Guarantee of a CSVC Network

To prove the delay guarantee of a network of CSVC servers in Theorem 5, we first prove the deadline guarantee of a single CSVC server (Theorem 4). In the following, a *non-preemptive* scheduling algorithm is one that does not preempt the transmission of a lower priority packet even after a higher priority packet arrives. On the other hand, a *preemptive* scheduling algorithm always ensures that the packet in service is the packet with the highest priority by possibly preempting the transmission of a lower priority packet. A non-preemptive algorithm is considered *equivalent* to a preemptive algorithm if the priority assigned to all the packets is the same in both. To simplify the proof of Theorem 4, we first state and prove the following Lemmas.

Lemma 6 *The* core virtual clock *of packet p_f^k satisfies*

$$VCore_{f,j}^k \;\geq\; VCore_{f,j}^{k-1} + \frac{l_f^k}{r_f}, \qquad k > 1$$

Proof: Observe that

$$\max_{i\in[1..k]} \frac{l_f^i}{r_f^i} \;\geq\; \max_{i\in[1..k-1]} \frac{l_f^i}{r_f^i} \tag{4.11}$$

Also observe that by repeatedly applying 4.10, we get:

$$\begin{aligned} VCore_{f,j}^{k-1} &= VCore_{f,2}^{k-1} + (j-2)\max_{i\in[1..k-1]} \frac{l_f^i}{r_f^i} + \sum_{i=2}^{j-1}(\beta_{f,i}+\pi_i) \\ &= VC_{f,1}^{k-1} + (j-1)\max_{i\in[1..k-1]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1}(\beta_{f,i}+\pi_i) \qquad (4.12) \end{aligned}$$

Similarly,

$$\begin{aligned} VCore_{f,j}^k &= VC_{f,1}^k + (j-1)\max_{i\in[1..k]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1}(\beta_{f,i}+\pi_i) \\ &\geq VC_{f,1}^{k-1} + \frac{l_f^k}{r_f} + (j-1)\max_{i\in[1..k]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1}(\beta_{f,i}+\pi_i) \qquad (from(4.6)) \\ &\geq VC_{f,1}^{k-1} + \frac{l_f^k}{r_f} + (j-1)\max_{i\in[1..k-1]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1}(\beta_{f,i}+\pi_i) \qquad (from(4.11)) \\ &\geq VCore_{f,j}^{k-1} + \frac{l_f^k}{r_f} \qquad (from(4.12)) \end{aligned}$$

■

Lemma 7 *If the j^{th} server's capacity is not exceeded, then the time at which packet p_f^k departs a Preemptive CSVC server, denoted by $L_{PCSVC}^j(p_f^k)$, is*

$$L_{PCSVC}^j(p_f^k) \;\leq\; VCore_{f,j}^k \qquad j \geq 1 \tag{4.13}$$

Proof:

At server j, define the quantity $R_{f,j}(t)$ for flow f as follows:

$$R_{f,j}(t) = \begin{cases} r^k_{f,j} & \text{if } \exists k \ni (a^k_{f,j} \le t) \wedge (VCore^{k-1}_{f,j} < t \le VCore^k_{f,j}) \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

Let S be the set of flows served by server j. Then server j with capacity C_j is defined to have *exceeded its capacity* at time t if $\sum_{n \in S} R_{n,j}(t) > C_j$.

The proof of Lemma 7 is by induction on j.

Base Case : j = 1

The first server in a CSVC network runs the Virtual Clock algorithm, for which the theorem is proved in [25].

Induction Hypothesis : Assume 4.13 holds for $1 \le j \le m$.

Induction : We will show that 4.13 holds for $1 \le j \le m+1$.

From (4.18) and the Induction Hypothesis (which implies that $g^k_{f,m} \ge 0$) we get

$$VCore^k_{f,m+1} \;\; \ge \;\; a^k_{f,m+1} + \frac{l^k_f}{r^k_f} \tag{4.15}$$

Let $K_f(t_1, t_2)$ be the set of all flow f packets that arrive in interval $[t_1, t_2]$ and have virtual clock value no greater than t_2. For packet p^k_f, let $T^k_{f,j}$ denote the following quantity

$$T^k_{f,j} \;\; = \;\; VCore^k_{f,j} - \max(a^k_{f,j}, VCore^{k-1}_{f,j})$$

It is easily observed from (4.14), (4.15) and Lemma 6 that

$$\int_{t_1}^{t_2} R_{f,m+1}(t)dt = \sum_{i \in K_f(t_1,t_2)} (r^i_{f,m+1} * T^i_{f,m+1}) \ge \sum_{i \in K_f(t_1,t_2)} (r^i_{f,m+1} * \frac{l^i_f}{r^i_f}) \ge \sum_{i \in K_f(t_1,t_2)} l^i_f$$

Therefore, the cumulative length of all flow f packets that arrive in interval $[t_1, t_2]$ and have virtual clock value no greater than t_2, denoted by $AP_f(t_1, t_2)$, is given as

$$AP_f(t_1, t_2) \;\; \le \;\; \int_{t_1}^{t_2} R_{f,j}(t)dt \tag{4.16}$$

We now prove the lemma by contradiction. Assume that for packet p_f^k, $L_{PCSVC}^{m+1}(p_f^k) > VCore_{f,m+1}^k$. Also, let t_0 be the beginning of the busy period in which p_f^k is served and $t_2 = VCore_{f,m+1}^k$. Let t_1 be the least time less than t_2 during the busy period such that no packet with virtual clock value greater than t_2 is served in the interval $[t_1, t_2]$. We claim that such a t_1 exists. If not, then either packet $p_{f,m+1}^k$ is the first packet in the busy period or preempts the service of the previous packet on arrival. In either case, the packet gets served immediately on arrival, and

$$L_{PCSVC}^{m+1}(p_f^k) \ = \ a_{f,m+1}^k + \frac{l_f^k}{C_{m+1}} \ \leq \ VCore_{f,m+1}^k$$

which violates our assumption. Therefore, such a t_1 exists.

Clearly, all the packets served in the interval $[t_1, t_2]$ arrive in this interval (else they would have been served earlier than t_1) and have virtual clock value less than or equal to t_2. Since the server is busy in the interval $[t_1, t_2]$ and packet p_f^k is not serviced by t_2, we have:

$$\begin{aligned}\sum_{f \in S} AP_f(t_1, t_2) &> C_{m+1}(t_2 - t_1) \\ \sum_{f \in S} \int_{t_1}^{t_2} R_{f,m+1}(t)dt &> C_{m+1}(t_2 - t_1) \\ \int_{t_1}^{t_2} \sum_{f \in S} R_{f,m+1}(t)dt &> C_{m+1}(t_2 - t_1) \qquad (4.17)\end{aligned}$$

Since the server capacity is not exceeded, $\sum_{f \in S} R_{f,m+1}(t) \leq C_{m+1}$. Hence, $\int_{t_1}^{t_2} \sum_{f \in S} R_{f,m+1}(t)dt \leq C_{m+1}(t_2 - t_1)$. This contradicts (4.17) and hence the induction step is proved. From induction, the lemma follows. ■

The following lemma is stated and proved in [25].

Lemma 8 *If PS is a work conserving preemptive scheduling algorithm, NPS its equivalent non-preemptive scheduling algorithm and the priority assignment of a packet is not changed dynamically, then*

$$L_{NPS}(p^k) - L_{PS}(p^k) \ \leq \ \frac{l^{max}}{C}$$

where $L_{PS}(p^k)$ and $L_{NPS}(p^k)$ denote the time a packet leaves the server when PS and NPS scheduling algorithms are employed, respectively. Also, l^{max} is the maximum length of a packet and C is the capacity of the server.

Theorem 4 *If a server's capacity is not exceeded, then the time at which packet p_f^k departs a Core Stateless Virtual Clock server, denoted by $L^j_{CSVC}(p_f^k)$, is*

$$L^j_{CSVC}(p_f^k) \leq VCore^k_{f,j} + \frac{l_j^{max}}{C_j}$$

where C_j is the capacity of the server and l_j^{max} is the maximum length of a packet serviced by server j.

Proof: As Preemptive CSVC algorithm is work conserving and does not dynamically change the priority of a packet, Theorem 4 follows immediately from Lemma 7 and Lemma 8. ■

Theorem 4 states that the deadline for the departure of packet p_f^k at server j is $(VCore^k_{f,j} + \beta_{f,j})$. To avoid errors in computation of $VCore^k_{f,j-1}$ and π_{j-1} due to inaccuracies in clock synchronization and imprecise knowledge of propagation delays, Equation (4.10) can be rewritten in terms of an alternate set of variables as:

$$VCore^k_{f,j} = a^k_{f,j} + g^k_{f,j-1} + \max_{i\in[1..k]} \frac{l_f^i}{r_f^i}, \qquad j \geq 2 \tag{4.18}$$

where $g^k_{f,j-1}$ is the amount of time by which p_f^k departs before its deadline, $(VCore^k_{f,j-1} + \beta_{f,j-1})$ at server $j-1$.

Theorem 5 *The delay guarantee of a network of CSVC servers is the same as that of a network of Virtual Clock servers.*

Proof: Using Theorem 4, Equation (4.9) and repeated application of Equation (4.10), it follows that the time at which packet p_f^k departs the last server in a network of H CSVC servers, $L^H_{CSVC}(p^k_f)$, is given by:

$$\begin{aligned}
L^H_{CSVC}(p^k_f) &\leq VCore^k_{f,H} + \beta_H \\
&\leq VC^k_{f,1} + (H-1)\max_{i\in[1..k]}\frac{l^i_f}{r^i_f} + \sum_{i=1}^{H-1}(\beta_{f,i}+\pi_i) + \beta_H \\
&\leq EAT_1(p^k_f) + \frac{l^k_f}{r_f} + (H-1)\max_{i\in[1..k]}\frac{l^i_f}{r^i_f} + \sum_{i=1}^{H-1}(\beta_{f,i}+\pi_i) + \beta_H
\end{aligned}$$

where $\beta_i = \frac{l^{max}_i}{C_i}$. The delay guarantee of a network of H Virtual Clock servers has been shown to be the same as the above in [24]. $\blacksquare$

4.4 Generalizing to CSGR Networks

In this section, we generalize the methodology described in the previous section to define a *core-stateless* version of any GR network, which gives the same delay guarantee as the original network.

We define the *core guaranteed rate clock* ($GRCore$) for packet p^k_f at server j, $GRCore^k_{f,j}$, as follows[1]:

$$\begin{aligned}
GRCore^k_{f,1} &= GRC^k_{f,1} + \beta_{f,1} && (4.19)\\
GRCore^k_{f,j+1} &= GRCore^k_{f,j} + \pi_j + \max_{i\in[1..k]}\frac{l^i_f}{r^i_f} + \beta_{f,j+1}, \quad j \geq 1 && (4.20)
\end{aligned}$$

For every GR algorithm, we define a corresponding *Core-Stateless GR* algorithm at servers $j > 1$, that assigns $GRCore$ values to packets of all flows as defined above, and schedules packets in the increasing order of their $GRCore$ values[2].

We derive the delay guarantee of a CSGR network below.

[1]This definition of $GRCore$ differs from the one given in [35]. The definition and analysis methodology given in [35] works only for GR networks in which $\beta_{f,j} \geq \frac{l^{max}_j}{C_j}$.

[2]Note that for CSVC as defined in Section 4.3, $VCore^k_{f,j} = GRCore^k_{f,j} - \beta_{f,j}$. Since $\beta_{f,j} = \frac{l^{max}_j}{C_j}$ is the same for every packet at server j, it can be seen that the packet transmission sequence—and consequently, the end-to-end delay bound—would remain the same if they are transmitted in the increasing order of their $GRCore$ values.

Lemma 9 *The* core guaranteed rate clock *of packet p_f^k satisfies*

$$GRCore_{f,j}^k \;\geq\; GRCore_{f,j}^{k-1} + \frac{l_f^k}{r_f}, \qquad k > 1$$

Proof: Observe that by repeatedly applying (4.20), we get:

$$\begin{aligned} GRCore_{f,j}^{k-1} &= GRCore_{f,1}^{k-1} + (j-1) \max_{i \in [1..k-1]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1} \pi_i + \sum_{i=2}^{j} \beta_{f,i} \\ &= GRC_{f,1}^{k-1} + (j-1) \max_{i \in [1..k-1]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1} (\beta_{f,i} + \pi_i) + \beta_{f,j} \qquad (4.21) \end{aligned}$$

Similarly,

$$\begin{aligned} GRCore_{f,j}^k &= GRC_{f,1}^k + (j-1) \max_{i \in [1..k]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1} (\beta_{f,i} + \pi_i) + \beta_{f,j} \\ &\geq GRC_{f,1}^{k-1} + \frac{l_f^k}{r_f} + (j-1) \max_{i \in [1..k]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1} (\beta_{f,i} + \pi_i) + \beta_{f,j} \\ &\geq GRC_{f,1}^{k-1} + \frac{l_f^k}{r_f} + (j-1) \max_{i \in [1..k-1]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1} (\beta_{f,i} + \pi_i) + \beta_{f,j} \\ &\geq GRCore_{f,j}^{k-1} + \frac{l_f^k}{r_f} \qquad (from (4.21)) \end{aligned}$$

■

Lemma 10 establishes the deadline guarantee of a CSGR server. Its proof methodology is based on the following insights. A CSGR server is a Delay-EDD server; the latter has been shown to have the largest schedulability region among all non-preemptive scheduling algorithms. If the corresponding GR algorithm was employed at the server, it would guarantee the departure of packet p_f^k by $GRCore_{f,j}^k$. It follows that a CSGR server, being a Delay-EDD server that transmits packets in the increasing order of their $GRCore$ values, would also provide the same guarantee.

Lemma 10 *If the schedulability conditions of the corresponding GR algorithm at the* j^{th} *server are met, then the time at which packet p_f^k departs a CSGR server,* *denoted by $L^j_{CSGR}(p^k_f)$, is given by:*

$$L^j_{CSGR}(p^k_f) \quad \leq \quad GRCore^k_{f,j} \qquad\qquad j \geq 1 \tag{4.22}$$

Proof: The proof of Lemma 10 is by induction on j.

Base Case : j = 1

The first server in a CSGR network is a GR server. By definition of the GR class, packet p^k_f departs server $j = 1$ by $GRC^k_{f,1} + \beta_{f,1} = GRCore^k_{f,1}$. Therefore, (4.22) holds at server $j = 1$.

Induction Hypothesis : Assume (4.22) holds for $1 \leq j \leq m$.

Induction : We will show that (4.22) holds for $1 \leq j \leq m + 1$.

From the Induction Hypothesis, we get: $a^k_{f,m+1} \leq GRCore^k_{f,m} + \pi_m$. From (4.20), therefore we get:

$$GRCore^k_{f,m+1} \quad \geq \quad a^k_{f,m+1} + \frac{l^k_f}{r_f} + \beta_{f,m+1} \tag{4.23}$$

From Lemma 9, we get:

$$GRCore^k_{f,m+1} \quad \geq \quad GRCore^{k-1}_{f,m+1} + \frac{l^k_f}{r_f} \tag{4.24}$$

We use the following claim to prove the induction step.

Claim 1: For all $k \geq 1$, $GRCore^k_{f,m+1} \geq GRC^k_{f,m+1} + \beta_{f,m+1}$.

Proof of Claim 1: We use induction on k to prove the claim for all $k \geq 1$.

Base Case : k = 1

$GRC^1_{f,m+1} = a^1_{f,m+1} + \frac{l^1_f}{r_f}$. From (4.23), therefore, $GRCore^1_{f,m+1} \geq GRC^1_{f,m+1} + \beta_{f,m+1}$.

Induction Hypothesis : Assume $GRCore^i_{f,m+1} \geq GRC^i_{f,m+1} + \beta_{f,m+1}$, for $1 \leq i \leq k - 1$.

Induction : We will show that the claim holds for $1 \leq i \leq k$.

Consider two cases for packet p^k_f:

Case 1: $a^k_{f,m+1} \geq GRC^{k-1}_{f,m+1}$. From (4.23), we get:

$$\begin{aligned} GRC^k_{f,m+1} &= a^k_{f,m+1} + \frac{l^k_f}{r_f} \\ &\leq GRCore^k_{f,m+1} - \beta_{f,m+1} \end{aligned}$$

Case 2: $a^k_{f,m+1} < GRC^{k-1}_{f,m+1}$. From the induction hypothesis and (4.24):

$$\begin{aligned} GRC^k_{f,m+1} &= GRC^{k-1}_{f,m+1} + \frac{l^k_f}{r_f} \\ &\leq GRCore^{k-1}_{f,m+1} + \frac{l^k_f}{r_f} - \beta_{f,m+1} \\ &\leq GRCore^k_{f,m+1} - \beta_{f,m+1} \end{aligned}$$

Hence, the induction step is proved, and it follows from induction that the claim is true for all $k \geq 1$.

The following observations complete the proof of the induction step:

Observation 1: If the corresponding stateful GR algorithm is used at server $m+1$, and its schedulability conditions are met, then for the same sequence of packet arrivals, it would guarantee that packet p^k_f would depart by $GRC^k_{f,m+1} + \beta_{f,m+1}$. From Claim 1, this also implies that the stateful GR algorithm guarantees that packet p^k_f would depart by $GRCore^k_{f,m+1}$.

Observation 2: Observe that:

$$\begin{aligned} GRCore^k_{f,j} &= GRC^k_{f,j} + (GRCore^k_{f,j} - GRC^k_{f,j}) \\ &= \max(a^k_{f,j}, GRC^{k-1}_{f,j}) + \frac{l^k_f}{r_f} + (GRCore^k_{f,j} - GRC^k_{f,j}) \end{aligned}$$

Since a CSGR server transmits packets in the increasing order of their $GRCore$ values, it is a Delay-EDD [20, 34, 67] server with $D^k_{f,j} = \frac{l^k_f}{r_f} + (GRCore^k_{f,j} - GRC^k_{f,j})$, where $D^k_{f,j}$ is the delay bound for p^k_f at server j [25].

Observation 3: It has been shown in [21] [3], that Delay-EDD has the largest schedulability region among all non-preemptive scheduling algorithms—this implies that if a GR algorithm at server $m+1$, which guarantees that p_f^k will depart by $GRCore_{f,m+1}^k$ (Observation 1), is replaced by a Delay-EDD scheduler that transmits packets in the increasing order of their $GRCore$ values, packet p_f^k would still be guaranteed to depart by $GRCore_{f,m+1}^k$.

From Observations 2 and 3, the induction step is proved. From induction, the lemma follows. ■

Theorem 6 *The delay guarantee of a network of CSGR servers is the same as that of a network of corresponding GR servers.*

Proof: From Lemma 10, the time at which packet p_f^k departs the last server in a network of H CSGR servers, $L_{CSGR}^H(p_f^k)$, is given by:

$$\begin{aligned} L_{CSGR}^H(p_f^k) &\leq GRCore_{f,j}^k \\ &= GRC_{f,1}^k + (H-1)\max_{i\in[1..k]} \frac{l_f^i}{r_f^i} + \sum_{j=1}^{H-1}(\beta_{f,j}+\pi_j) + \beta_{f,H} \\ &= EAT_1(p_f^k) + \frac{l_f^k}{r_f} + (H-1)\max_{i\in[1..k]} \frac{l_f^i}{r_f^i} + \sum_{j=1}^{H-1}(\beta_{f,j}+\pi_j) + \beta_{f,H} \end{aligned}$$

The delay guarantee of a corresponding network of H GR servers has been shown to be the same as the above in [24]. ■

From Theorem 6, it follows that for every GR algorithm, we can define a *Core-Stateless GR* algorithm that would preserve the delay properties of the original GR algorithm. Of particular interest is the design of Core-Stateless-Delay-EDD, a work-conserving GR algorithm that has the desirable property of *decoupling* the delay and rate guarantees [17, 63].

[3]The result in [21] has been shown only for flows where the inter-arrival time of packets is at least $\frac{l_f^k}{r_f}$. However, in a networking environment, inter-arrival times may become smaller than that. It has been shown in [25], that regardless of the inter-arrival times, Delay-EDD guarantees the departure of packets by their deadline.

4.5 Summary

In this work, we propose a methodology for transforming any *Guaranteed Rate* (GR) network into its core-stateless counterpart, that eliminates the need for maintaining per-flow state or performing per-packet classification in core routers. Our methodology is based on the key insight that: *upper bounds on packet deadlines at any core node can be computed using per-flow state at the edge node.* We demonstrate that a CSGR network, that uses the upper bounds on deadlines, instead of actual deadlines, provides the same end-to-end delay guarantee. Since the GR class is fairly general, this methodology provides a tool to design a wide range of core-stateless networks that provide delay guarantees.

It is worthwhile to note that core-stateless networks that provide end-to-end delay guarantees, while being work-conserving, have also been presented in [40, 66]. Core-stateless networks with other desirable properties such as decoupling the delay and rate guarantees have also been proposed in [66], independent of our work.

Observe that the method for deriving core-stateless versions of GR algorithms described above preserves *only* the delay property of the GR algorithms. Hence, this approach can be used to derive core-stateless versions of all unfair, work-conserving algorithms (such as Virtual Clock, Delay EDD) in GR. However, if this approach is applied to algorithms in GR that provide other forms of guarantees (e.g., fairness), then the resulting core-stateless algorithm may not provide the same type of guarantees. For instance, if the above technique is applied to the Weighted Fair Queuing (WFQ) [18] algorithm, then the resulting core-stateless algorithm will have the same delay property as WFQ, but would not guarantee fairness. The design of core-stateless networks that provide throughput and fairness properties is the subject matter of subsequent chapters in this dissertation.

Chapter 5

Core-stateless Guaranteed Throughput Networks

In this chapter, we design a core-stateless network that provides end-to-end throughput guarantees at short time-scales, similar to those provided by core-stateful networks. We achieve this objective in two steps. First, we show that for a network to provide throughput guarantees, it *must* also provide end-to-end delay guarantees. Second, we demonstrate that two mechanisms —tag re-use and source rate control— when integrated with a core-stateless network that provides end-to-end delay guarantees, lead to the design of the Core-stateless Guaranteed Throughput (CSGT) networks that provide end-to-end throughput bounds within an *additive constant* of that attained by a core-stateful network of fair servers. Throughout this chapter, we assume that a source transmits equal-sized packets.

5.1 Need for Delay Guarantees

The following theorem establishes that delay guarantees are necessary for providing throughput guarantees.

Theorem 7 *If a network provides lower-bounds on throughput of the form:* $W_{f,H}(t_1, t_2) \geq r_f(t_2 - t_1) - \Phi$ *to any flow* f *whose source transmits at least at its reserved rate, then it also provides to flow* f *an end-to-end delay guarantee of the form:* $d^k_{f,H} - EAT_1(p^k_f) \leq \frac{\Phi + l^{max}_f}{r_f}$.

Proof: Consider $t_1 = a^1_{f,1}$ and $t_2 = d^k_{f,H}$. Then $W_{f,H}(a^1_{f,1}, d^k_{f,H}) = \sum_{i=1}^{k} l^i_f$. Substituting into the throughput bound, we get: $d^k_{f,H} \leq a^1_{f,1} + \frac{\Phi}{r_f} + \sum_{i=1}^{k} \frac{l^i_f}{r_f}$. Since the source transmits at least at its reserved rate, $EAT_1(p^k_f) = a^1_{f,1} + \sum_{i=1}^{k-1} \frac{l^i_f}{r_f}$. Therefore, $d^k_{f,H} - EAT_1(p^k_f) \leq \frac{\Phi + l^k_f}{r_f} \leq \frac{\Phi + l^{max}_f}{r_f}$, for all k. Therefore, the network provides a delay guarantee to flow f. ■

The converse of Theorem 7 indicates that *a network that does not provide delay guarantee to packets cannot provide throughput bounds.* Hence, a work-conserving, core-stateless network that provides delay guarantee is a crucial building block for designing CSGT networks.

CSGR networks, described in Chapter 4, are core-stateless networks that provide the same delay guarantee as a network that employs stateful GR scheduling algorithms at the core routers. As we illustrate below, a work-conserving CSGR network, however, does not provide throughput guarantees at short time-scales. Our design of a CSGT network uses the CSGR network as a building block and enhances it with a set of end-to-end mechanisms that allow the network to retain its delay properties while providing throughput guarantees at short time-scales. We describe the derivation of a CSGT network in the context of a Core-stateless Virtual Clock (CSVC) network [35, 66]—a specific instance of the class of CSGR networks.

5.2 Defining CSGT Networks

Throughput in CSVC Networks We have previously shown that a CSVC network (Section 4.3) provides a *deadline guarantee*: packet p^k_f is guaranteed to depart server j by $(VCore^k_{f,j} + \beta_{f,j})$. However, the following example shows that a CSVC network does not provide throughput guarantees at finite time-scales.

Example 1 *Consider the first server, with a transmission capacity of* 10 *packets/sec, in a CSVC network. Let the sum of reserved rates of all flows be equal to the capacity. Let the rate reserved by a flow f be* 1 *packet/sec. At time $t = 0$, let f be the only backlogged flow. In this setting, by $t = 1$, 10 packets of flow f are serviced by the server; further, $VCore^{11}_{f,1} = 11$. Now, let all other flows become backlogged at time $t = 1$. Since the server services packets in the increasing order of virtual clock values, packet p^{11}_f may not be serviced until $t = 10$; hence, flow f receives no throughput during the interval* $[2, 10]$. *Given any time interval of arbitrary length, it is easy to extend this example to show that flow f receives no throughput during the interval of interest. Therefore, for any interval length, the CSVC server does not provide any non-trivial (non-zero) lower bound on throughput.*

In the above example, until time $t = 1$, because of the availability of idle bandwidth and the work-conserving nature of the CSVC server, flow f receives service at a rate greater than its reserved rate. Due to the way deadlines are computed, though, during the same period, flow f accumulates a *debit* at the rate r_f (indicated by the increase in its VCore value much beyond current time t), and is subsequently penalized for the duration of the accumulated debit once all the other flows become backlogged. It is important for networks to provide throughput guarantees at short time-scales, independent of the past usage of idle bandwidth by a flow, for two reasons:

1. In many settings, is is difficult for sources to predict precisely their bandwidth requirements at short time-scales. For instance, the bit-rate requirement of a variable bit-rate video stream may vary considerably and over short time-scales. At the onset of transmission, therefore, it may be difficult to predict

accurately the rate to reserve—sometimes the video stream may use less than its reserved rate, sometimes it may use more. Suppose a video stream, with a reserved bit-rate of $1Mbps$, transmits at $2Mbps$ for 4 seconds using idle bandwidth. In a network that does not provide throughput guarantees, the video stream may not receive any throughput at all in the next 4 seconds. The performance of the video application in such a network may therefore be unacceptable.

2. It is in the best interest of a network to allow sources to transmit data in transient bursts (i.e., at a rate greater than the reserved rate); bursty transmissions allow a network to benefit from statistical multiplexing of the available network bandwidth among competing traffic. In networks that penalize sources for using idle bandwidth, however, sources have no incentive to transmit bursts into the network. They may prefer to use constant bit-rate flows, instead of allowing the network to enforce arbitrary penalties. This, in turn, would reduce the statistical multiplexing gains and thereby reduce the overall utilization of network resources.

It is important to observe that while a CSVC network does not provide lower bounds on throughput at finite time-scales, it does guarantee an *average* throughput at the rate of r_f to a backlogged flow f over infinite time-scales. This implies that, the throughput of flow f in any interval $[t_1, t_2]$ would be below its reserved rate r_f only if the flow f receives service at a rate higher than r_f prior to t_1. In such an event, there must exist $t', t" < t_1$ such that during interval $[t', t"]$, packets of flow f arrive at the destination much ahead of their deadline guarantee (derived based on the reserved rate r_f). More formally, for packets p_f^k that reach destination at time t during the interval $[t', t"]$, $VCore_{f,H}^k \gg t$.

The Principle The property of allowing a flow to accumulate arbitrarily large amount of debit —by increasing the deadline values (or service tag values) assigned to packets of the flow much beyond the current time— is central to the inability of CSVC networks to provide throughput guarantees at small time-scales. Hence, for a network to provide throughput bounds at small time-scales, it must reduce debit accumulation; this can be achieved by *allowing the ingress routers to re-use for future packets the deadline (or service tag) values of packets that reach the destination much prior to their deadlines.* This is the central concept in transforming a CSVC network into a CSGT network that provides throughput bounds.

The Definition of CSGT Network A CSGT network, like the CSVC network, consists of two types of routers: *edge routers* and *core routers.* The ingress edge router, in addition to maintaining per-flow state, maintains a sorted-list $\mathcal{R}$ of re-usable tag vectors. On receiving a packet p_f^k of flow f, the ingress router assigns to it a *service tag vector* $[F_1(p_f^k), F_2(p_f^k), ..., F_H(p_f^k)]$ where H is the number of servers along the path, and $F_j(p_f^k)$ is the service tag for server j. The assignment of the tag vector to packet p_f^k proceeds as follows: If $\mathcal{R} \neq \emptyset$, an incoming packet is assigned the smallest tag vector from $\mathcal{R}$. Otherwise, a new tag vector is created as follows:

$$F_1(p_f^k) \quad = \quad \max(a_{f,1}^k, \widehat{F}(a_{f,1}^k)) + \frac{l_f^k}{r_f} \tag{5.1}$$

$$F_j(p_f^k) \quad = \quad F_1(p_f^k) + \sum_{h=1}^{j-1}(\beta_{f,h} + \pi_h + \max_{1 \leq i \leq k} \frac{l_f^i}{r_f}), \; j > 1 \tag{5.2}$$

where $\beta_{f,h} = \frac{l_h^{max}}{C_h}$, and $\widehat{F}(t)$ is the maximum of the service tags for server 1 assigned to any packet by time t. All servers in the CSGT network transmit packets in the increasing order of their service tags for that server.

Observe that if $\mathcal{R} = \emptyset$, then the assignment of tag vector in CSGT is identical to the CSVC network. When $\mathcal{R} \neq \emptyset$, then, by reusing a tag assigned to an earlier packet, CSGT prevents accumulation of unbounded debit for flow f. To instantiate such a CSGT network, we need to address the following issues.

1. When can an ingress server reuse a previously assigned tag vector for a new packet? What are the constraints that govern the re-usability of tag vectors? How does the ingress router create and maintain the sorted-list $\mathcal{R}$? We address these questions in Section 5.2.1.
2. With the reuse of previously assigned tag vectors in the CSGT network, packets of flow f with higher sequence number may, in fact, carry a smaller tag value (i.e., $F_h(p_f^j) < F_h(p_f^i)$ even if $i < j$). Since the tag values determine the priority for servicing packets in each router, it is quite possible that packet p_f^j may reach the egress edge router prior to packet p_f^i, even though packet p_f^i was transmitted prior to packet p_f^j at server 1. We discuss the associated packet re-ordering requirement in Section 5.2.2.

5.2.1 Maintaining the Sorted-list of Re-usable Tag Vectors

Reusing tag vectors allow CSGT networks to prevent unbounded debit accumulation for flows. Determination of whether a tag vector is eligible for reuse, however, is tricky because of two reasons.

- A CSGT network must ensure that the reuse of tag vectors for packets of flow f does not violate the deadline guarantees provided to *other* flows.

  To meet this requirement, the tag assigned to a packet p_f^k must differ by at least $\frac{l_f}{r_f}$ from the tags assigned to all packets p_f^i that were transmitted prior to packet p_f^k but have not reached the destination. This is because, if the separation is less than $\frac{l_f}{r_f}$, then flow f will be *guaranteed* service at a rate greater than its reserved rate r_f; this, in turn, could violate the deadline guarantees provided to other flows.
- A CSGT network must ensure that it can provide a deadline guarantee on the re-used tag vector.

  To meet this requirement, at the time of assigning a re-usable tag to a packet, the ingress router must ensure that the tag value for the first server exceeds the current time by at least $\frac{l_f}{r_f}$.

Using these eligibility criteria, we formally define re-usability of a tag vector as follows.

Definition 8 *A previously assigned tag vector* $[F_1, F_2, ..., F_H]$ *is said to be* **re-usable** *for a packet* p_f^m *at time* t *if it satisfies the following properties:*

$$\forall p_f^i \in \mathcal{U} : |F_j - F_j(p_f^i)| \geq \frac{l_f}{r_f} \tag{5.3}$$

$$t \leq F_1 - \frac{l_f}{r_f} \tag{5.4}$$

where $\mathcal{U}$ *is the set of packets transmitted by server 1 prior to packet* p_f^m *but have not reached the destination by time* t.

A CSGT network can enforce these conditions as follows.

1. An ingress router should consider a tag vector for re-use only after a packet carrying that tag vector departs the egress router H. This ensures that condition (5.3) is met. This can be achieved by requiring the egress router to send, on transmitting a packet p_f^m of flow f, an acknowledgment for that packet to the ingress router for flow f. The ingress router, on receiving such an acknowledgment, can add the tag vector assigned to packet p_f^m to the sorted-list $\mathcal{R}$ of re-usable tag vectors for flow f.
2. On receiving a packet p_f^k from flow f at time t, the ingress router can scan through the sorted-list $\mathcal{R}$, discard all the tag vectors that violate condition (5.4), and assign to packet p_f^k the first re-usable tag that meets condition (5.4).

Observe that the tag vector assigned to a packet p_f^m is likely to re-usable (i.e., satisfy condition (5.4)) only if packet p_f^m departs server H "sufficiently" prior to its deadline. In particular, if D^{min} is the minimum latency incurred by the

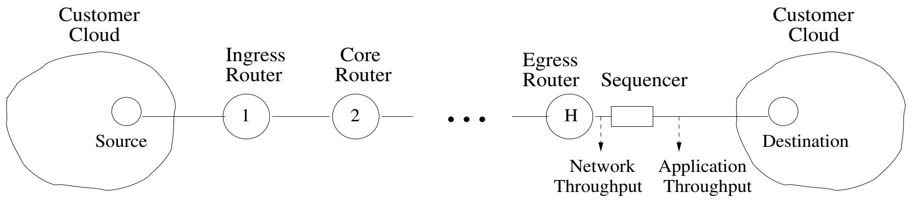


Figure 5.1: The CSGT Network Architecture

acknowledgment packet to reach the ingress router, then using (5.4), the tag vector of packet p_f^m can be re-used only if: $d_{f,H}^m + D^{min} \leq F_1(p_f^m) - \frac{l_f}{r_f}$. From (5.2), this is the same as:

$$d_{f,H}^m \quad \leq \quad F_H(p_f^m) - \left(\sum_{j=1}^{H-1} (\beta_{f,j} + \pi_j + \max_{1 \leq i \leq m} \frac{l_f^i}{r_f}) + D^{min} + \frac{l_f^{min}}{r_f} \right) \tag{5.5}$$

Thus, the egress server sends an acknowledgment to the first server, *only* if packet p_f^k departs the network much before—as given by condition (5.5)—its deadline. We prove, in Lemma 13 (see Section 5.3), that if a CSGT network reuses tag vectors in accordance with the scheme described above, then it provides the same deadline-guarantee as a CSVC network.

5.2.2 Addressing Packet Re-ordering Requirements

With the tag re-use scheme described above, in a CSGT network, packets of flow f may reach the egress router out-of-order. For applications that desire in-order delivery semantics, a CSGT network needs to employ a *sequencer* that can buffer packets received out-of-order and then deliver to the applications packets in-order. A sequencer can reside either on the egress router, on a special network appliance located between the egress edge router and the destination node, or on the destination node itself[1]. Figure 5.1 depicts the setting where a sequencer is logically inserted between the egress router and the destination node. For the simplicity of analysis,

[1]Deploying a sequencer on the destination node itself may require changes to end-hosts. Hence, the architectural options of instantiating the sequencer on the egress router or on an appliance located at the edge of the customer network may be more desirable.

we assume zero propagation delay between the egress router and the sequencer.

Now, let us consider the issues in designing the sequencer. The following example shows that, in a naive implementation of a CSGT network, the number of packets that may need to be buffered at the sequencer is not bounded.

Example 2 *Consider the case when the tag vector of packet p_f^k becomes re-usable at the source, there are n unacknowledged packets, $p_f^{k+1}, \ldots, p_f^{k+n}$, with* **larger** *tag vectors in the network. Let the tag vector of p_f^k be re-assigned to packet p_f^{k+n+1}. Now let the tag vectors of the first $(n-1)$ unacknowledged packets $p_f^{k+1}, \ldots, p_f^{k+n-1}$ also become available for re-use; let these tag vectors be assigned to subsequent $(n-1)$ packets, namely, $p_f^{k+n+2}, \ldots, p_f^{k+n+n}$. Consider the case where packet p_f^{k+n} departs the egress node at its deadline, $F_H(p_f^{k+n})$. Since packets $p_f^{k+n+1}, \ldots, p^{k+n+n}$ have smaller deadlines, they are guaranteed to depart the egress router earlier than p_f^{k+n}. Therefore, these n packets need to be buffered, simultaneously for some time, at the sequencer till packet p_f^{k+n} arrives. Larger the value of n, the larger the buffer space requirement at the sequencer.*

In practice, a sequencer would have a fixed amount of buffer space. In order to avoid packet loss due to overflow of the sequencer buffers, therefore, the aggressiveness of sources using a CSGT network may need to be controlled. We do this by employing a *flow control algorithm* that limits the maximum number of deadlines that are simultaneously in use for packets of a flow. Specifically, the flow control algorithm ensures that at any point in time t, no packet is assigned a deadline larger than $t + W\frac{l_f}{r_f}$, where W is a configuration parameter. When a packet arrives at time t, if no deadline smaller than $t + W\frac{l_f}{r_f}$ is available for assigning to it, the packet is held till one is available.

Observe that a large value of W increases the buffer space requirement at the sequencer (Example 2). A small value of W, on the other hand, limits the extent to which the source can utilize idle bandwidth in the network. In fact, if $W = 1$, the first server does not transmit a packet before its expected arrival time; in this case the server reduces to the non-work-conserving Jitter Virtual Clock server. In practice, the largest value of W—such that buffer overflow at the sequencer can be avoided—should be selected. If B denotes the available sequencer buffer space, in units of the packet size l_f, then Lemma 11 provides a condition that when satisfied by W, avoids packet loss due to buffer overflow.

Lemma 11 *Packets of flow f will not be dropped at the sequencer due to unavailability of re-ordering buffers if W satisfies:*

$$B \geq \begin{cases} (N+1)(W-1) - (N)(N+1)\frac{k^{min}}{2}, & \text{if } T^{min} \geq \frac{l_f}{r_f} \\ \frac{W(W-1)}{2k^{min}}, & \text{if } T^{min} < \frac{l_f}{r_f} \end{cases} \tag{5.6}$$

where $N = \left\lfloor \frac{W-2}{k^{min}} \right\rfloor$, $k^{min} = \frac{T^{min}}{l_f/r_f}$, and T^{min} is a lower bound on the round-trip time[2]*.*

Proof: We refer to packets that are assigned re-used tags as *future* packet. We assume that a future packet is removed from the sequencer re-ordering buffers as soon as all packets with smaller sequence numbers arrive.

Let $B(t)$ denote the occupancy of the sequencer re-ordering buffers of a flow f at time t. Let t_i denote the time instant at which the i^{th} future packet is removed from these buffers. For ease of analysis, we assume that even if all packets with smaller sequence numbers reach the sequencer before it, a future packet is still buffered and removed (in such as scenario, at time instances t_i^- and t_i respectively). Let t_0 denote the initial time at which the source starts transmitting packets, and $B(t_0) = 0$. It follows that, $B(t_i) \leq B(t_i^-) - 1$ (more than one future packets may be removed from the buffers at the same instant). The buffer occupancy in any time interval $[t_i, t_{i+1})$ is non-decreasing with time (since no packets are removed in

[2]*T^{min} is a lower bound on the time difference between the transmission of a packet at the first server and the arrival of its acknowledgment at the first server. For instance, the sum of propagation and minimum transmission latencies on all the links on both the forward and reverse path qualifies as a lower bound.*

this interval). Therefore, the maximum buffer occupancy in this interval is given by $B(t_{i+1}^-)$.

Consider the first server (ingress node) at a time t_i. Consider all future packets transmitted by this server, that have not departed the sequencer (by time t_i). Of these, let b_1 denote the future packet with the smallest sequence number. Consider the set of all packets with smaller sequence numbers than packet b_1, that have not yet (at t_i) reached the sequencer. Among these, let p' have the largest tag-vector—then all of these packets would reach the sequencer at most by time $(F_H(p') + \beta_{f,H})$, and the packet b_1 would not need to be buffered after that.

Let t' be the time at which packet p' arrives at the first server, and is assigned a tag-vector. Since all future packets that have not departed the sequencer by time t_i, have a larger sequence number than p', they are transmitted from the first server after t'. Let B" be the total number of future packets that get transmitted from the first server in the interval $(t', F_1(p')]$, with *smaller* tag-vectors than p'. Since p' has not reached the sequencer by t_i, any future packets transmitted after t' with *larger* tag-vectors than p' have also not reached the sequencer. Hence, $B(t_i) \leq B$", and B" is the maximum number of future packets that would need to be buffered at the sequencer before p' gets delivered, that is, $B(t_{i+1}^-) \leq B$".

Due to source flow control, observe that: $t' \geq F_1(p') - W * l_f/r_f$. The number of distinct tag-vectors that lie in the interval $(t', F_1(p')]$ is given by: $W' = \left\lfloor \frac{(F_1(p'))^- - t'}{l_f/r_f} \right\rfloor \leq W - 1$. Let T^{min} denote a lower bound on the round-trip time—the time difference between the transmission of a packet at the first server and the arrival of its acknowledgment at the first server. The maximum number of tags that can become re-usable from the interval $(t', F_1(p')]$ after time t', which is an upper bound on $B(t_{i+1}^-)$, is given by:

$$B(t_{i+1}^-) \quad \leq \quad \begin{cases} W' + (W' - \lceil k^{min} \rceil) + ... + (W' - \lceil N'k^{min} \rceil), & \text{if } T^{min} \geq \frac{l_f}{r_f} \\ \left\lfloor \frac{1}{k^{min}} \right\rfloor + \left\lfloor \frac{2}{k^{min}} \right\rfloor + \ldots + \left\lfloor \frac{W'}{k^{min}} \right\rfloor, & \text{if } T^{min} < \frac{l_f}{r_f} \end{cases}$$

where $k^{min} = \frac{T^{min}}{l_f/r_f}$, and $N' = \left\lfloor \frac{W'-1}{k^{min}} \right\rfloor$. The right-hand-side of the above is an increasing function of W'. Since $W' \leq W - 1$,

$$B(t_{i+1}^-) \leq \begin{cases} (N+1)(W-1) - (N)(N+1)\frac{k^{min}}{2}, & \text{if } T^{min} \geq \frac{l_f}{r_f} \\ \frac{W(W-1)}{2k^{min}}, & \text{if } T^{min} < \frac{l_f}{r_f} \end{cases}$$

where $N = \left\lfloor \frac{W-2}{k^{min}} \right\rfloor$. Since this upper bound on $B(t_{i+1}^-)$ is independent of i, it is an upper bound on the maximum buffer occupancy in all time intervals $[t_i, t_{i+1}), i \geq 0$. Therefore, if the provisioned buffer space, B, is at least as large as given by this upper bound, no packets are lost at the sequencer re-ordering buffers. ■

Given the largest value of W that satisfies (5.6), there is a bound on the maximum amount of available bandwidth that a flow f can utilize. Conversely, one can provision buffer space at the sequencer that allows a flow to utilize up to a maximum bandwidth (say r'). In Appendix B, we show that to allow a source to utilize bandwidth r', the chosen value of W should satisfy the following condition[3]:

$$W \geq \frac{r'}{l_f}\left(\sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + D^{max}\right) + H + 1 \qquad (5.7)$$

Such a value of W can then be used to provision the sequencers with the appropriate amount of buffers. In particular, given r', the maximum bandwidth that a flow should be allowed to utilize, one can derive a bound on W using (5.7); this value of W when substituted in (5.6) determines the minimum buffer requirement at the sequencer.

5.3 Properties of CSGT Networks

5.3.1 Delay Guarantee

Lemma 12 establishes the deadline guarantee of a preemptive CSGT server.

Lemma 12 *If the j^{th} server's capacity is not exceeded, then the time at which packet*

[3]The condition in (5.7) can be intuitively seen to be a form of the commonly used *delay-bandwidth product* rule-of-thumb.

p_f^k *departs a Preemptive CSGT server, denoted by* $L^j_{PCSGT}(p_f^k)$*, is*

$$L^j_{PCSGT}(p_f^k) \quad \leq \quad F_j(p_f^k) \qquad\qquad j \geq 1 \tag{5.8}$$

Proof: Let $S_j(p_f^k) = F_j(p_f^k) - l_f^k/r_f$. At server j, define the quantity $R_{f,j}(t)$ for flow f as follows:

$$R_{f,j}(t) \quad = \quad \begin{cases} r_f & \text{if } \exists k \ni (a^k_{f,j} \leq t) \wedge (S_j(p_f^k) < t \leq F_j(p_f^k)) \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

Let S be the set of flows served by server j. Then server j with capacity C_j is defined to have *exceeded its capacity* at time t if $\sum_{n\in S} R_{n,j}(t) > C_j$. Let $K_{f,j}(t_1, t_2)$ be the set of all flow f packets that arrive at server j in interval $[t_1, t_2]$ and have deadlines no greater than t_2. For packet p_f^k, let $T^k_{f,j} = F_j(p_f^k) - \max(a^k_{f,j}, S_j(p_f^k))$. The proof of Lemma 12 is by induction on j.

<u>Base Case</u> : j = 1. From (5.1) and (5.4), we have: $F_1(p_f^k) \geq a^k_{f,1} + l_f^k/r_f^k$. Then it can be observed from (5.3) and (5.9) that:

$$\int_{t_1}^{t_2} R_{f,1}(t)dt \;=\; \sum_{i\in K_f(t_1,t_2)} (r^i_{f,1} * T^i_{f,1}) \;\geq\; \sum_{i\in K_f(t_1,t_2)} (r^i_{f,1} * \frac{l^i_f}{r^i_f}) \;\geq\; \sum_{i\in K_f(t_1,t_2)} l^i_f$$

Therefore, the cumulative length of all flow f packets that arrive in interval $[t_1, t_2]$ and have deadline value no greater than t_2, denoted by $AP_f(t_1, t_2)$, is given as $AP_f(t_1, t_2) \leq \int_{t_1}^{t_2} R_{f,1}(t)dt$.

We now prove the lemma by contradiction. Assume that for packet p_f^k, $L^1_{PCSGT}(p_f^k) > F_1(p_f^k)$. Also, let t_0 be the beginning of the busy period in which p_f^k is served and $t_2 = F_1(p_f^k)$. Let t_1 be the least time less than t_2 during the busy period such that no packet with deadline value greater than t_2 is served in the interval $[t_1, t_2]$. We claim that such a t_1 exists. If not, then either packet $p^k_{f,1}$ is the first packet in the busy period or preempts the service of the previous packet on arrival. In either case, the packet gets served immediately on arrival, and

$$L^1_{PCSVC}(p_f^k) \;=\; a^k_{f,1} + \frac{l_f^k}{C_1} \;\leq\; VCore^k_{f,1}$$

which violates our assumption. Therefore, such a t_1 exists.

Clearly, all the packets served in the interval $[t_1, t_2]$ arrive in this interval (else they would have been served earlier than t_1) and have deadline value less than or equal to t_2. Since the server is busy in the interval $[t_1, t_2]$ and packet p_f^k is not serviced by t_2, we have: $\sum_{f \in S} AP_f(t_1, t_2) > C_1(t_2 - t_1)$. Since $AP_f(t_1, t_2) \leq \int_{t_1}^{t_2} R_{f,1}(t)dt$, we have:

$$\int_{t_1}^{t_2} \sum_{f \in S} R_{f,1}(t)dt \quad > \quad C_1(t_2 - t_1) \tag{5.10}$$

Since the server capacity is not exceeded, $\sum_{f \in S} R_{f,1}(t) \leq C_1$. Hence, $\int_{t_1}^{t_2} \sum_{f \in S} R_{f,1}(t)dt \leq C_1(t_2 - t_1)$. This contradicts (5.10) and hence the base case is proved.

Induction Hypothesis : Assume (5.8) holds for $1 \leq j \leq m$.

Induction Step : We will show that (5.8) holds for $1 \leq j \leq m + 1$.

From (5.2) and the Induction Hypothesis, we get: $F_{m+1}(p_f^k) \geq a_{f,m+1}^k + l_f^k / r_f^k$. The induction step can now be proved in exactly the same manner as the base case. Therefore, from induction, the lemma follows. ■

The following lemma proves that deadline guarantees of CSVC are preserved in a CSGT network.

Lemma 13 *A packet p_f^k is guaranteed to depart server j in a CSGT network by $(F_j(p_f^k) + \beta_{f,j})$.*

Proof: Since Preemptive CSGT is work conserving and does not dynamically change the priority of a packet, lemma 13 follows immediately from Lemma 12 and Lemma 8.
■

5.3.2 Throughput Guarantee

To quantify the effect of packet re-ordering on the throughput received by the applications, we define two different throughput measures. We define *network throughput* as the number of bits that depart the egress router during a given time interval, and *application throughput* as the number of bits that depart the sequencer (after re-ordering) during the interval. Note that the application throughput in any given interval may be different from the network throughput. Theorem 8 provides lower bounds on the network and application throughput in a CSGT network.

Theorem 8 *If the source of flow f transmits packets at least at its reserved rate, and D^{max} is an upper bound on the latency after which an acknowledgment packet sent by the egress node reaches the ingress node, then the network guarantees a minimum throughput in any time interval $[t_1, t_2]$, $W_{f,H}(t_1, t_2)$, given by:*

$$W_{f,H}(t_1,t_2) \quad > \quad r_f(t_2 - t_1) - r_f\left((H+2)\frac{l_f}{r_f} + \sum_{j=1}^{H-1}\pi_j + \sum_{j=1}^{H}\beta_{f,j}\right) - r_f * D^{max}$$

Further, the sequencer guarantees a minimum throughput, $W_f^{app}(t_1, t_2)$, given by:

$$W_f^{app}(t_1,t_2) \quad > \quad r_f(t_2 - t_1) - r_f\left((H+1)\frac{l_f}{r_f} + \sum_{j=1}^{H-1}\pi_j + \sum_{j=1}^{H}\beta_{f,j}\right) - r_f * D^{max} - W * l_f$$

Proof: Since D^{max} is the maximum latency after which a packet from the egress node reaches the ingress node, the following condition is *sufficient* to ensure that the tags of packet p_f^k will be reused by the ingress node:

$$d_{f,H}^k + D^{max} \quad \leq \quad F_1(p_f^k) - \frac{l_f^k}{r_f} \tag{5.11}$$

Consider any tag value F at the *last* server H. Let k_F denote the sequence number of the <u>last</u> packet that is ever assigned the deadline tag F at server H. Then the departure time of packet $p_f^{k_F}$ from the last server, $d_H^{k_F}$, must not satisfy condition (5.11)—otherwise, tag F would be re-used for another packet, and $p_f^{k_F}$ does not qualify to be the last packet to be assigned the tag F. We therefore have:

$$F - d_H^{k_F} \quad < \quad D^{max} + \frac{l_f^{k_F}}{r_f^{k_F}} + \sum_{j=1}^{H-1}(\beta_{f,j} + \pi_j + \max_{1\leq i\leq k_F}\frac{l_f^i}{r_f})$$

Let $\widehat{T}$ be the upper bound on the right hand side of the above for all F. That is, for a source with equal-sized packets, let $\widehat{T} = D^{max} + H\frac{l_f}{r_f} + \sum_{j=1}^{H-1}\pi_j + \sum_{j=1}^{H-1}\beta_{f,j}$.

Now consider any time interval (t_1, t_2). From the definition of $\widehat{T}$, we know that given any $F \geq (t_1 + \widehat{T})$ that has been assigned as a deadline tag at the last server H, at least one packet with this tag F will be delivered after time t_1 (this is true even for $t_1 = a^1_{f,1}$, since $F_1(p^1_f) \leq a^1_{f,1} + \widehat{T}$). Further, from Lemma 13, we know that any packet with an assigned deadline tag F will be delivered no later than $F + \beta_{f,H}$. Therefore, for *every* $F \in [t_1 + \widehat{T}, t_2 - \beta_{f,H}]$ that has been assigned as a tag to any packet, *at least* one packet with that deadline tag will depart the last server in the time interval (t_1, t_2). Let $B(t_0, t)$ represent the total number of bits in packets that are *last* assigned deadlines that lie in any time interval $(t_0, t]$. Then, $W_{f,H}(t_1, t_2) \geq B(t_1 + \widehat{T}, t_2 - \beta_{f,H})$.

Let $t_3 = t_1 + \widehat{T}$ and $t_4 = t_2 - \beta_{f,H}$. Let $t'_3 \geq t_3$ be the smallest time instant that coincides with a deadline—let the corresponding packet be $p^{k_3}_f$. Let $t'_4 \leq t_4$ be the largest time instant that coincides with a deadline—let the corresponding packet be $p^{k_4}_f$. Then, since the source transmits at least at its reserved rate, the total number of bits in packets with deadline in the range (t'_3, t'_4) is given by: $B(t'_3, t'_4) = r_f(t'_4 - t'_3)$.

Now note that $t'_3 < t_3 + l_f/r_f$. This is so because otherwise, $F(p^{k_3-1}_f) \in [t_3, t'_3)$, which violates the definition of t'_3. Similarly, $t'_4 > t_4 - l_f/r_f$. This is so because otherwise, $F(p^{k_4+1}_f) \in (t'_4, t_4]$, which violates the definition of t'_4. Therefore, we have: $t'_4 - t'_3 > (t_4 - t_3) - 2l_f/r_f$. Therefore, we get: $W_{f,H}(t_1, t_2) \geq B(t_3, t_4) \geq B(t'_3, t'_4) = r_f(t'_4 - t'_3) > r_f(t_4 - t_3) - 2l_f$. This implies,

$$W_{f,H}(t_1, t_2) \quad > \quad r_f(t_2 - t_1) - r_f\left((H+2)\frac{l_f}{r_f} + \sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j}\right) - r_f * D^{max}$$

Next, consider a time instant $t_5 \in (t_3, t_4)$ that coincides with a deadline tag at the last server. Let $p^{k_5}_f$ be the last packet that is assigned the deadline t_5. Then $p^{k_5}_f$ is received at the sequencer in the interval (t_1, t_2). If $p^{k_5}_f$ is not a future packet, it departs the sequencer (for simplicity, we assume that packets depart the sequencer instantly). If not, it has to be buffered till all packets with smaller sequence numbers reach the sequencer as well. Among all these packets with smaller sequence numbers than $p_f^{k_5}$, let p' have the largest deadline (if there are more than one such packets, let p' be the last packet to be assigned that deadline).

Packet $p_f^{k_5}$ arrives for transmission at the first server latest by $F_1(p_f^{k_5}) - l_f^{k_5}/r_f$. Let t' be the time at which p' arrives for transmission at the first server. Since p' has a smaller sequence number than $p_f^{k_5}$, $t' < F_1(p_f^{k_5}) - l_f^{k_5}/r_f$. Further, due to source flow control, $F_1(p') \leq t' + W * l_f/r_f$. Therefore, $F_1(p') < F_1(p_f^{k_5}) + (W-1) * l_f/r_f \Rightarrow F_H(p') < t_5 + (W-1) * l_f/r_f$.

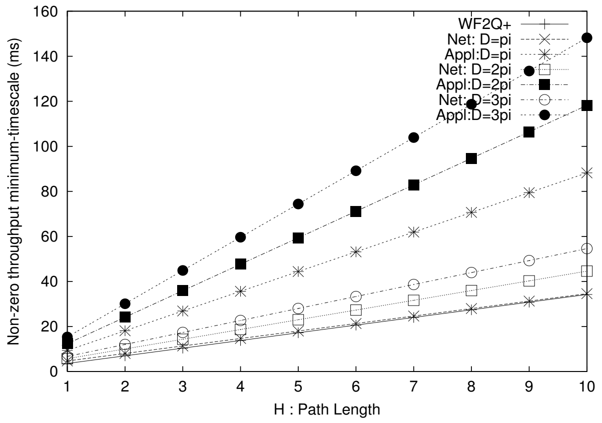
Since p' is guaranteed to be delivered at the sequencer by $F_H(p') + \beta_{f,H}$, $p_f^{k_5}$ will also be delivered by this time. This implies that, for any deadline in the interval $(t_1 + \widehat{T}, t_2 - \beta_{f,H} - (W-1) * l_f/r_f)$, the last packet to be assigned that deadline will depart *the sequencer* in the time interval (t_1, t_2). Therefore, the number of bits that depart the sequencer in the time interval (t_1, t_2), $W_f^{app}(t_1, t_2)$, is given by: $W_f^{app}(t_1, t_2) \geq B(t_3, t_4 - (W-1) * \frac{l_f}{r_f}) \geq B(t_3', t_4' - (W-1) * \frac{l_f}{r_f}) > r_f(t_4 - t_3) - 2r_f\frac{l_f}{r_f} - (W-1)l_f$. This implies,

$$W_f^{app}(t_1, t_2) \quad > \quad r_f(t_2 - t_1) - r_f\left((H+1)\frac{l_f}{r_f} + \sum_{j=1}^{H-1}\pi_j + \sum_{j=1}^{H}\beta_{f,j}\right) - r_f * D^{max} - W * l_f$$

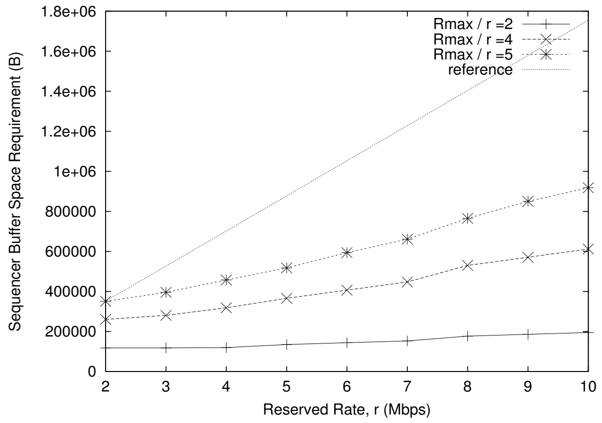
■

The bound on the network throughput derived in Theorem 8 for a CSGT network differs from that provided by a core-stateful network (Theorem 2), by a constant term $E_1 = r_f * [D^{max} - \sum_{j=1}^{H}(\epsilon_{f,j} - \beta_{f,j})] + l_f$. The bound on application throughput differs by the additional term $E_2 = (W-1) * l_f$.

Observe that for a CSGT network derived from CSVC, $\beta_{f,j} = l^{max}/C_j$. Further, for most stateful schedulers, $\epsilon_{f,j} \geq l^{max}/C_j$. Therefore, $E_1 \leq r_f * D^{max} + l_f$, which is primarily governed by D^{max}, the maximum latency on the reverse path.



(a) Non-zero throughput time-scale



(b) Sequencer buffer space requirement

Figure 5.2: Evaluation of a CSGT Network

5.4 Evaluation of CSGT Networks

5.4.1 The Throughput Guarantee

Theorem 8 states that when measured over any time interval larger than $\gamma_{f,H}^{net}$, the network guarantees a non-zero throughput at a rate equal to the reserved rate. As discussed in Section 3.2, the smaller the value of $\gamma_{f,H}^{net}$, the better the network can support applications with stringent timeliness requirements. To evaluate the throughput guarantee of a CSGT network numerically, we compute $\gamma_{f,H}^{net}$ for example networks, where link capacities are $100Mbps$ and link propagation latencies are $1ms$. In figure 5.2(a), we plot $\gamma_{f,H}^{net}$ against the number of hops on the end-to-end path of a sample flow with a reserved bit-rate of $10Mbps$. D^{max} is varied from a multiple of 1 to 3 of the end-to-end propagation latency on the reverse path. For comparison, we also plot $\gamma_{f,H}^{net}$ for a representative core-stateful network of fair WF^2Q+ [7] servers. We observe the following:

1. When D^{max} is equal to the end-to-end link propagation latency, the throughput guarantee of the CSGT networks is similar to that of the core-stateful WF^2Q+ networks.

2. $\gamma_{f,H}^{net}$ increases with D^{max}. Therefore, the throughput guarantee of a CSGT network improves by provisioning low-delay feedback channels. However, even when D^{max} is three times the end-to-end propagation latency, the application is guaranteed a non-zero throughput over any time interval larger than $150ms$.

These observations imply that by provisioning low-delay feedback channels, a CSGT network can provide non-zero throughput guarantees at very short time-scales, and similar to those in core-stateful networks.

5.4.2 Sequencer Buffer Space vs. Maximum Throughput

Recall that there is a tradeoff between the amount of buffer space required at the sequencer and the maximum rate that the source is allowed to achieve. We numerically characterize this tradeoff for the same example network as above (link capacity $= 100Mbps$, link propagation latency $= 1ms$). In figure 5.2(b), we plot the minimum sequencer buffer space required to allow sources to achieve a given maximum bit-rate, R^{max} (varied from 2 to 5 times the reserved rate). We observe that:

1. To enable a flow, with a reserved bit-rate of up to $10Mbps$, to achieve an end-to-end bit-rate of up to 5 times that ($50Mbps$), less than $1MB$ of sequencer buffer space is sufficient.
2. The buffer space requirement grows slower with increase in the bit-rate of the sample flow (for reference, we plot a line where the buffer requirement grows at the same rate as the reserved bit-rate).

These observations imply that a small amount of sequencer buffer space is sufficient to allow flows to utilize bandwidth up to multiple times their reserved rate. Further, a CSGT network can reduce the total buffer requirement at the sequencer by aggregating into a single large flow, all micro-flows that traverse the same path between a pair of edge routers.

5.5 Summary

In this chapter, we present the Core-stateless Guaranteed Throughput (CSGT) network architecture—the *first* work-conserving network architecture that provides throughput guarantees to individual flows over finite time-scales, but without maintaining per-flow state in core routers. We develop the architecture in two steps. First, we show that for a network to provide end-to-end throughput guarantees, it must also provide end-to-end delay guarantees. Second, we demonstrate that two mechanisms —tag re-use and source rate control— when integrated with a work-conserving, core-stateless network that provides end-to-end delay guarantees, lead to the design of CSGT network that provides end-to-end throughput bounds within an *additive constant* of what is attained by a core-stateful network of fair rate servers.

Chapter 6

Core-stateless Guaranteed Fair Networks

Fairness guarantees differ from delay or throughput guarantees in a fundamental way; unlike delay and throughput guarantees, that can be characterized entirely in terms of the intrinsic properties of a flow (such as its reserved rate), the definition of a fairness guarantee is inherently a function of the state of all other flows sharing a resource. In this chapter, we explore the design of networks that provide fairness guarantees without maintaining per-flow state in core routers. Our design proceeds in two steps. First, we show that for a network to provide fairness guarantees, it must also provide throughput guarantees. Second, we explore mechanisms that, when integrated with a CSGT network that provides throughput guarantees, lead to the design of the *Core-stateless Guaranteed Fair* (CSGF) network architecture—the *first* work-conserving core-stateless architecture that provides deterministic fairness guarantees.

6.1 The Need for End-to-end Throughput Guarantees

Theorem 9 indicates that providing throughput guarantees is necessary for providing fairness guarantees in work-conserving networks.

Theorem 9 *A work-conserving server that provides fairness guarantees to a continuously-backlogged flow m of the form:*

$$\frac{W_{f,j}(t_1, t_2)}{r_f} \leq \frac{W_{m,j}(t_1, t_2)}{r_m} + U_{j,\{f,m\}} \tag{6.1}$$

where f is any other flow (not necessarily backlogged), also provides to flow m a throughput guarantee of the form:

$$W_{m,j}(t_1, t_2) \geq r_m(t_2 - t_1) - r_m * \gamma_{m,j}$$

where $\gamma_{m,j} = \frac{l^{max}}{C_j} + \frac{\sum_{f \in F} r_f U_{j,\{f,m\}}}{C_j}$.

Proof: A work-conserving server that has at least one continuously backlogged flow in $[t_1, t_2]$ would satisfy:

$$\sum_{f \in F} W_{f,j}(t_1, t_2) \geq C_j(t_2 - t_1) - l^{max}$$

where the l^{max} term appears due to packetization effects.

Consider a particular interval $[t_1, t_2]$. Let $a = \frac{W_{m,j}(t_2 - t_1)}{r_m}$. From (6.1), for any other flow f (whether continuously backlogged or not), we have: $W_{f,j}(t_2 - t_1) \leq r_f * a + r_f * U_{j,\{f,m\}}$. Therefore,

$$\begin{aligned} C_j(t_2 - t_1) - l^{max} &\leq \sum_{f \in F} W_{f,j}(t_1, t_2) \\ &\leq a * \sum_{f \in F} r_f + \sum_{f \in F} r_f U_{j,\{f,m\}} \\ &\leq a * C_j + \sum_{f \in F} r_f U_{j,\{f,m\}} \end{aligned}$$

This implies,

$$a \;\geq\; (t_2 - t_1) - \frac{l^{max}}{C_j} - \sum_{f \in F} \frac{r_f U_{j,\{f,m\}}}{C_j}$$

Therefore, $W_{m,j}(t_1, t_2) \geq r_m(t_2 - t_1) - r_m \gamma_{m,j}$, where $\gamma_{m,j} = \frac{l^{max}}{C_j} + \frac{\sum_{f \in F} r_f U_{j,\{f,m\}}}{C_j}$.
■

The converse of the above theorem states that *a server that does not provide throughput guarantees cannot provide fairness guarantees.* A core-stateless network that provides throughput guarantees is, therefore, a crucial building block for the design of one that provides fairness guarantees. Note, however, that a network that provides throughput guarantees need not be fair. For instance, a network may provide throughput exactly at the reserved rate to one flow, but may allow another flow to use significantly more than its reserved rate.

A simple scheme for providing fairness in a network that provides throughput guarantees is to employ a set of two mechanisms: (1) treat the aggregate traffic between a pair of edge routers as a single flow and provide throughput guarantees to it, and (2) employ a fair scheduling algorithm at the ingress node that allocates a fair share of the aggregate throughput to individual flows within the aggregate. With such a scheme, packet departures at the last node in any time interval $[t_1, t_2]$ can be equated to packet departures at the ingress node in some other time interval $[t'_1, t'_2]$; the end-to-end fairness guarantee of the network is, therefore, exactly the same as the fairness guarantee of the ingress server.

To design a core-stateless network architecture that provides fairness guarantees, we use the above set of mechanisms in conjunction with a CSGT architecture, that provides throughput guarantees. The resultant architecture—referred to as a *Core-stateless Guaranteed Fair (CSGF)* network—is defined below.

Definition of a CSGF network: The ingress router for F, a set of flows sharing the same end-to-end path in a CSGF network, has two logical component:

- **Deadline Assignment:** A packet that belongs to the "aggregate" flow is assigned a tag-vector exactly as in a CSGT network; new tag-vectors are computed using a reserved rate of $R = \sum_{f \in F} r_f$.
- **Packet Selection:** The next packet to be assigned a deadline is selected according to a fair schedule of transmission across flows in F. We use a Start-time Fair Queuing (SFQ) [26] scheduler to determine the next flow to select a packet from. SFQ has the desirable property of fair allocation even when the available capacity is fluctuating, which is the case in a dynamic network environment.

The core routers and the egress router in a CSGF network function in the same manner as in a CSGT network. The "aggregate" is split into micro-flows at the egress; for traffic classes that require in-order delivery semantics, a sequencer re-orders packets before they are split into micro-flows.

In Section 6.2, we derive the fairness guarantees of a CSGT network. In the rest of this chapter, we assume that all flows between the same pair of edge routers transmit packets of the same size l[1].

6.2 Fairness Guarantees in a CSGF network

Our objective is to compute $\frac{W_{f,H}(t_1,t_2)}{r_f} - \frac{W_{m,H}(t_1,t_2)}{r_m}$, the difference in normalized throughput received by two backlogged flows f and m during time interval $[t_1, t_2]$. Recall that two kinds of throughput can be defined in a CSGT (or a CSGF) network— the *application* throughput and the *network* throughput—depending on whether it is measured after packets are reordered by the sequencer (if at all), or before. Correspondingly, fairness in a CSGF network can be defined with respect

[1]If l^{max} is the maximum allowed packet size, it is reasonable to expect a source that is backlogged with data to transmit, to use packets of size l^{max}.

to either the application or the network throughput. For service classes that guarantee an in-order delivery semantics to applications, end-to-end fairness applies to the application throughput. For applications that do not require in-order delivery semantics—such as large file transfers— however, it is important to characterize fairness in network throughput as well.

6.2.1 Fairness in Application Throughput

Let p_1 and p_2, respectively, be the first and last packets belonging to the aggregate that depart the sequencer during a time interval $[t_1, t_2]$. Then, since packets depart the sequencer in increasing order of deadlines, all and only packets transmitted between p_1 and p_2 at the ingress node, depart the sequencer during $[t_1, t_2]$. The unfairness measure guaranteed to flows for the *application throughput* during $[t_1, t_2]$ in a CSGF network is, therefore, *equal* to that of the SFQ server at the ingress node during some other time interval. For any two backlogged flows f and m, it follows that:

$$\frac{W_f^{app}(t_1, t_2)}{r_f} \leq \frac{W_m^{app}(t_1, t_2)}{r_m} + U_{f,m}^{SFQ} \tag{6.2}$$

6.2.2 Fairness in Network Throughput

It may be tempting to conclude that the fairness guarantee on the *network throughput* received by two flows in a CSGF network also has the same fairness guarantees as the fair ingress server. This is, however, not the case. This is so because the underlying assumption behind equating the fairness of the network to the fairness of the ingress server is that packets depart the network in the same order as transmitted at the ingress server. In a CSGT network, however, packets may get re-ordered in the network due to deadline re-use. As a result, packet departures at the last node in any time interval $[t_1, t_2]$ may not be equal to packet departures at the ingress node in any other time interval. For instance, if p_1 is the first packet to depart the last node during the time interval $[t_1, t_2]$, then some packets transmitted after p_1 at the first node (with smaller deadlines than p_1), may depart the last node before t_1. Similarly, some packets transmitted before p_1 at the first node (with larger deadlines than p_1), may depart the last node after t_1. So the natural question is: *how exactly is the end-to-end fairness guarantee on network throughput related to that of the ingress server?* We answer this question in the rest of this section. In the following, let $T_h(p)$ denote the departure time of packet p from node h.

Let $n_f^+(p, t)$ denote the number of packets of flow f transmitted *after* packet p at the first node, which depart the last node *before* time t. Let $n_f^-(p, t)$ denote the number of packets of flow f transmitted *before* packet p at the first node, which depart the last node *after* time t. Let p_1' and p_2' denote the packet with the smallest deadline to depart the last node after time t_1 and t_2, respectively. Then, it follows that:

$$\begin{aligned} W_{f,H}(t_1, t_2) \;\; &= \;\; W_{f,1}(T_1(p_1'), T_1(p_2')) + \; l * n_f^-(p_1', t_1) - l * n_f^+(p_1', t_1) \\ & \quad - \; l * n_f^-(p_2', t_2) + l * n_f^+(p_2', t_2) \end{aligned}$$

Therefore, end-to-end fairness can be computed using:

$$\begin{aligned} \frac{W_{f,H}(t_1, t_2)}{r_f} - \frac{W_{m,H}(t_1, t_2)}{r_m} \;\; &= \;\; \frac{W_{f,1}(T_1(p_1'), T_1(p_2'))}{r_f} - \frac{W_{m,1}(T_1(p_1'), T_1(p_2'))}{r_m} \\ & \quad + l * \left(\frac{n_m^+(p_1', t_1)}{r_m} - \frac{n_f^+(p_1', t_1)}{r_f} \right) \\ & \quad - l * \left(\frac{n_m^-(p_1', t_1)}{r_m} - \frac{n_f^-(p_1', t_1)}{r_f} \right) \\ & \quad - l * \left(\frac{n_m^+(p_2', t_2)}{r_m} - \frac{n_f^+(p_2', t_2)}{r_f} \right) \\ & \quad + l * \left(\frac{n_m^-(p_2', t_2)}{r_m} - \frac{n_f^-(p_2', t_2)}{r_f} \right) \end{aligned}$$

To compute the unfairness measure, we need to compute an upper bound on the right hand side of the above relation. For computing the $n_m^+, n_m^-, n_f^+, n_f^-$ terms, we assume that the number of packets transmitted from flow f and m during $[T_1(p'_1), F_1(p'_1)]$ and $[T_1(p'_2), F_1(p'_2)]$ are equal. If not, then the term $\frac{W_{f,1}(T_1(p'_1),T_1(p'_2))}{r_f} - \frac{W_{m,1}(T_1(p'_1),T_1(p'_2))}{r_m}$ captures the difference.

Computing maximum $\left(\frac{n^-_m(p'_1,t_1)}{r_m} - \frac{n^-_f(p'_1,t_1)}{r_f}\right)$: Let $\delta_1 = \frac{F_1(p'_1)-T_1(p'_1)}{l/R}$. Of the packets transmitted at the first node before p'_1, only those with deadlines larger than p'_1 may depart the last node after t_1 (follows from the definition of p'_1). If p'_1 is the first packet to be transmitted with the deadline $F_1(p'_1)$, then no packets with larger deadlines could have been transmitted before p'_1. If p'_1 is the n^{th} packet to be transmitted with the deadline $F_1(p'_1)$, then the $(n-1)^{th}$ such packet departs the last node before t_1 (follows from the definition of p'_1). It follows that of the packets transmitted at the first node before p'_1 with larger deadlines, only those may depart the last node after t_1, which were transmitted after such an $(n-1)^{th}$ packet with deadline $F_1(p'_1)$. The maximum number of such packets of flow m, is given by: $n^-_m(p'_1, t_1) \leq W - \delta_1$. The minimum number of such packets of flow f is given by: $n^-_f(p'_1, t_1) \geq 0$. Therefore,

$$\left(\frac{n^-_m(p'_1, t_1)}{r_m} - \frac{n^-_f(p'_1, t_1)}{r_f}\right) \quad \leq \quad \frac{W - \delta_1}{r_m} \tag{6.3}$$

Computing maximum $\left(\frac{n^+_f(p'_1,t_1)}{r_f} - \frac{n^+_m(p'_1,t_1)}{r_m}\right)$: Any packet transmitted from the first node after p'_1, and assigned at least as large a deadline as p'_1, would depart the last node after t_1 (since p'_1, which departs after t_1, can not overtake such packets). Therefore, $n^+_f(p'_1, t_1)$ is the total number of packets of flow f transmitted from the first node after p'_1, and assigned a smaller deadline than p'_1[2].

These packets are transmitted during $[T_1(p'_1), F_1(p'_1) - \frac{l}{R}]$ (since a packet transmitted after $F_1(p'_1) - \frac{l}{R}$ can not be assigned a deadline smaller than $F_1(p'_1)$). The throughput of the first node during the interval $[T_1(p'_1), F_1(p'_1) - \frac{l}{R}]$ satisfies the

[2]Note from the definition of p'_1, that all such packets depart before t_1.

fairness property—that is, the difference in number of packets transmitted during $[T_1(p'_1), F_1(p'_1) - \frac{l}{R}]$ from flow f and m is bounded. Of these packets of flow f and m, some may be transmitted with smaller deadlines than $F_1(p'_1)$, and some with larger. $\left(\frac{n_f^+(p'_1,t_1)}{r_f} - \frac{n_m^+(p'_1,t_1)}{r_m}\right)$ is maximized when the following conditions are met:

1. *Packets with smaller deadlines predominantly belong to flow f.*

   The transmission of packets from the first node is interleaved across flows (due to the fair schedule of transmission). Therefore, the above condition is satisfied when deadlines assigned to sequentially-transmitted packets are also interleaved between those that are smaller than $F_1(p'_1)$ (assigned to packets of flow f) and those that are larger (assigned to packets of flow m).

2. *The number of packets of flow f (and therefore, flow m) is maximized.*

   The number of flow f packets get maximized when (i) f and m are the only flows backlogged at the first node, so that the fair-share throughput of each is at its maximum, and (ii) deadlines get re-used the maximum number of times.

At time $T_1(p'_1)$, deadline-reuse is maximized when δ_1 packets with deadlines smaller than $F_1(p'_1)$ and $W - \delta_1$ packets with larger deadlines are transmitted. Then, among packets transmitted with smaller deadlines, the normalized number of packets of flow f (or m) may exceed the number of packets of flow m (or f) by at most $\min\left(\frac{\lfloor\delta_1\rfloor}{r_f}, \frac{W-\lceil\delta_1\rceil}{r_m}\right)$.

The next set of packets are transmitted no earlier than at time $T_1(p'_1)+T^{min}$, where T^{min} is the minimum latency after which the acknowledgment for a packet can arrive at the first node. At time $T_1(p'_1) + T^{min}$, at most $\delta_1 - k^{min}$ packets with deadlines smaller than $F_1(p'_1)$ and at most $W-\delta_1+k^{min}$ packets with larger deadlines can be transmitted, where $k^{min} = \frac{T^{min}}{l/R}$. Again, among packets transmitted with smaller deadlines, the number of packets of flow f may exceed the number of packets of flow m by at most $\min\left(\frac{\lfloor\delta_1-k^{min}\rfloor}{r_f}, \frac{W-\lceil\delta_1-k^{min}\rceil}{r_m}\right)$.

By repeating the above construction of packet transmission for the complete duration of the time interval $[T_1(p'_1), F_1(p'_1) - \frac{l}{R}]$, we find that the total number of packets of flow f transmitted during this interval with deadlines smaller than $F_1(p'_1)$, may exceed those of flow m by up to:

$$\begin{aligned}\left(\frac{n_f^+(p'_1,t_1)}{r_f} - \frac{n_m^+(p'_1,t_1)}{r_m}\right) &\leq \min\left(\frac{\lfloor\delta_1\rfloor}{r_f}, \frac{W - \lceil\delta_1\rceil + 1}{r_m}\right) \\ &+ \min\left(\frac{\lfloor\delta_1 - k^{min}\rfloor}{r_f}, \frac{W - \lceil\delta_1 - k^{min}\rceil + 1}{r_m}\right) + \ldots \\ &+ \min\left(\frac{\lfloor\delta_1 - Nk^{min}\rfloor}{r_f}, \frac{W - \lceil\delta_1 - Nk^{min}\rceil + 1}{r_m}\right)\end{aligned}$$

where $N = \left\lfloor \frac{\delta_1 - 1}{k^{min}} \right\rfloor$.

Computing an upper bound on $\left(\frac{n_f^-(p'_1,t_1)}{r_f} - \frac{n_m^-(p'_1,t_1)}{r_m}\right) + \left(\frac{n_m^+(p'_1,t_1)}{r_m} - \frac{n_f^+(p'_1,t_1)}{r_f}\right)$**:**
Since $\delta_1 \in [0, W]$, the above term is maximized for $\delta_1 = W$, and reduces to:

$$\begin{aligned}&= \frac{1 + (\lfloor k^{min}\rfloor + 1) + \ldots + (\lfloor \widehat{n}k^{min}\rfloor + 1)}{r_f} + \frac{(W - \lceil(\widehat{n}+1)k^{min}\rceil) + \ldots + (W - \lceil Nk^{min}\rceil)}{r_m} \\ &\leq \frac{1 + (k^{min} + 1)\ldots + (\widehat{n}k^{min} + 1)}{r_f} + \frac{(W - (\widehat{n}+1)k^{min}) + \ldots + (W - Nk^{min})}{r_m} \\ &\leq \frac{(\widehat{n}+1)}{r_f} + k^{min}\frac{\widehat{n}(\widehat{n}+1)}{2r_f} + \frac{(N - \widehat{n})W}{r_m} - k^{min} * \left(\frac{N(N+1)}{2r_m} - \frac{\widehat{n}(\widehat{n}+1)}{2r_m}\right) \\ &\leq \frac{(\widehat{n}+1)}{r_f} + k^{min}\frac{\widehat{n}(\widehat{n}+1)}{2}\left(\frac{1}{r_f} + \frac{1}{r_m}\right) + \frac{(N - \widehat{n})W}{r_m} - k^{min} * \left(\frac{N(N+1)}{2r_m}\right)\end{aligned}$$

where $N = \lfloor \frac{W-1}{k^{min}} \rfloor$, and $\widehat{n}$ is such that: $\frac{W-2}{2k^{min}} - 1 < \widehat{n} < \frac{W}{2k^{min}}$, which is satisfied by $\widehat{n} = \left\lfloor \frac{W}{2k^{min}} \right\rfloor$. It can be seen that $-1 \leq \frac{N}{2} - \widehat{n} \leq 1$. Therefore, the above term reduces to:

$$\begin{aligned}&\leq \frac{(\widehat{n}+1)}{r_f} + k^{min}\frac{\widehat{n}(\widehat{n}+1)}{2}\left(\frac{1}{r_f} + \frac{1}{r_m}\right) + \frac{(\widehat{n}+2)W}{r_m} - k^{min}\frac{(\widehat{n}-1)(2\widehat{n}-1)}{r_m} \\ &\leq \frac{(\widehat{n}+1)}{r_f} + k^{min}\frac{\widehat{n}(\widehat{n}+1)}{2}\left(\frac{1}{r_f} + \frac{1}{r_m}\right) + \frac{(\widehat{n}+2)W}{r_m}\end{aligned}$$

Computing $\left(\frac{n_f^-(p_2',t_2)}{r_f} - \frac{n_m^-(p_2',t_2)}{r_m}\right) + \left(\frac{n_m^+(p_2',t_2)}{r_m} - \frac{n_f^+(p_2',t_2)}{r_f}\right)$: By repeating the above analysis for packet transmissions around packet p_2', we get:

$$\left(\frac{n_f^-(p_2',t_2)}{r_f} - \frac{n_m^-(p_2',t_2)}{r_m}\right) + \left(\frac{n_m^+(p_2',t_2)}{r_m} - \frac{n_f^+(p_2',t_2)}{r_f}\right)$$
$$\leq \frac{(\widehat{n}+1)}{r_f} + k^{min}\frac{\widehat{n}(\widehat{n}+1)}{2}(\frac{1}{r_f} + \frac{1}{r_m}) + \frac{(\widehat{n}+2)W}{r_m}$$

Computing the Unfairness Measure The total unfairness measure is therefore given by:

$$\frac{W_{f,H}(t_1,t_2)}{r_f} - \frac{W_{m,H}(t_1,t_2)}{r_m}$$
$$\leq U_{f,m}^{SFQ} + 2(\widehat{n}+1)\frac{l}{r_f} + k^{min}\widehat{n}(\widehat{n}+1)(\frac{l}{r_f} + \frac{l}{r_m}) + 2\frac{(\widehat{n}+2)Wl}{r_m} \quad (6.4)$$

where $\widehat{n} = \left\lfloor \frac{W}{2k^{min}} \right\rfloor$, $k^{min} = \frac{T^{min}}{l/R}$, and T^{min} is the minimum latency after which the acknowledgment for a packet can arrive at the first node.

It follows from (6.4), that a CSGF network provides deterministic fairness guarantees to flows on network throughput as well.

6.2.3 How Good Are CSGF Fairness Guarantees?

The CSGF is the *first* work-conserving core-stateless network architecture that provides deterministic end-to-end fairness guarantees. We next address the question: *how do the fairness guarantees of a CSGF network compare to those provided by core-stateful fair networks?* To answer this question, we compute the value of the unfairness measure for an example network topology where the link capacities are $100Mbps$ and the end-to-end link propagation latency on a 10-hop path is $10ms$.

Fairness in Application Throughput Observe that the fairness guarantee on application throughput in a CSGF network (given in (6.2)) is even better than that provided by core-stateful networks (derived in Theorem 3). The reason for this is

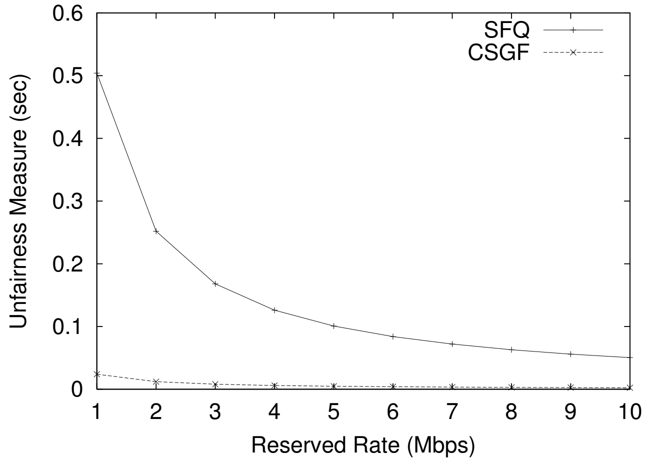
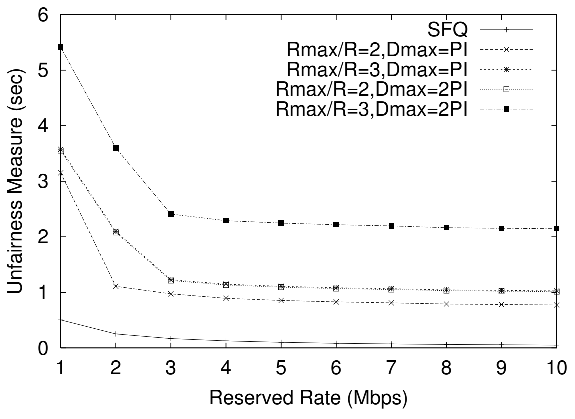


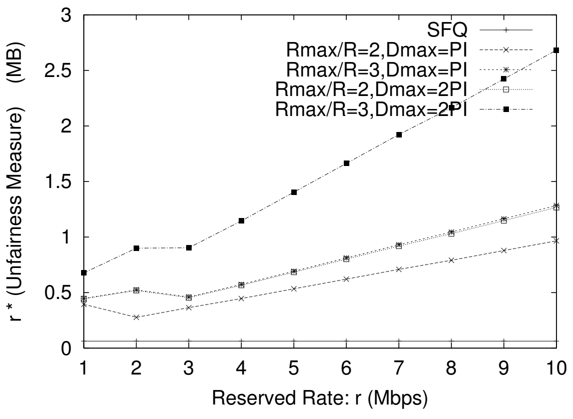
Figure 6.1: Fairness Guarantee on Application Throughput in a CSGF Network

that while packets depart the sequencer in a CSGF network exactly in the same order as transmitted by the fair ingress server, packets from different flows that share the same end-to-end path in a core-stateful network may not depart the network in the same order. The end-to-end fairness guarantee of a core-stateful network, therefore, can not be equated to that of its ingress server. Figure 6.1 plots the unfairness measure in a CSGF network as well as a core-stateful network of SFQ servers [26], each with the topology configuration described above, for different values of the reserved rates of micro-flows (assuming $r_f = r_m$). We observe that, for large-scale network topologies, the unfairness measure in a core-stateful architecture can be an order of magnitude higher than in a CSGF architecture.

Fairness in Network Throughput Observe that the unfairness measure on network throughput in (6.4) is a function of W and the bit-rates of the flows under consideration, r_f and r_m. Recall from Chapter 5 that W is a function of R^{max}, the maximum rate the aggregate traffic between the pair of edge routers is allowed to achieve, and D^{max}, the maximum latency experienced by acknowledgments on the reverse path. In Figure 6.2(a), therefore, we vary the reserved rate for the micro-



(a) Unfairness Measure



(b) Maximum Difference in Throughput

Figure 6.2: Fairness Guarantee on Network Throughput in a CSGF Network

flows (assuming $r_f = r_m$), and plot the unfairness measure for different values of R^{max}/R and D^{max}. For reference, we also plot the unfairness measure for a core-stateful network of SFQ [26] servers. We observe that:

1. A CSGF network provides a smaller unfairness measure when throughput received by flows with larger reserved rates are compared.

2. The unfairness measure increases when the aggregate traffic between a pair of edge routers is allowed to utilize bandwidth multiple times the total reserved rate for the aggregate.

3. The unfairness measure decreases with reduction in D^{max}. The fairness guarantee of a CSGF network, therefore, improves by provisioning low-delay feedback channels. However, even when D^{max} is equal to the end-to-end link propagation latency (Π), the unfairness measure in the CSGF network is an order of magnitude larger than in a core-stateful SFQ network.

Figure 6.2(a) indicates that the throughput received in a CSGF network by two flows f and m, during a given time interval, may differ by an amount worth playing out for a few seconds. To put this observation in perspective, we plot the reserved rate multiplied by the unfairness measure, in Figure 6.2(b). We observe that, during a given time interval, the network may deliver a few mega-bytes of more data for one flow, in comparison to other flows.

To summarize, the fairness guarantees on application throughput provided by a CSGF network are even better than those provided by core-stateful networks; fairness guarantees on network throughput are, however, weaker.

6.3 Throughput Guarantees in a CSGF Network

Theorem 8, which derives the application and network throughput guarantees of a CSGT network, indicates that the *aggregate* traffic between the pair of edge routers in a CSGF network is guaranteed a minimum throughput characterized by $R = \sum_{f \in F} r_f$, the cumulative reserved rate. Since each flow gets a fair share of this throughput, it follows from Theorem 9, that each micro-flow is provided a throughput guarantee as well.

Application Throughput Guarantee The application throughput guarantee of a CSGT network derived in Theorem 8, when applied to a CSGF network that re-orders packets, implies that

$$W_F^{app}(t_1, t_2) > R(t_2 - t_1) - R\left((H+1)\frac{l}{R} + \sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + D^{max}\right) - W * l \quad (6.5)$$

Let $a = \frac{W_m^{app}(t_1,t_2)}{r_m}$. From (6.5) and (6.2), it follows that:

$$\begin{aligned} R(t_2 - t_1) - R\left((H+1)\frac{l}{R} + \sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j}\right) - R * D^{max} - W * l & \\ \leq \sum_{f \in F} W_f^{app}(t_1, t_2) & \\ \leq \sum_{f \in F} a * r_f + r_f * U_{f,m}^{SFQ} & \end{aligned}$$

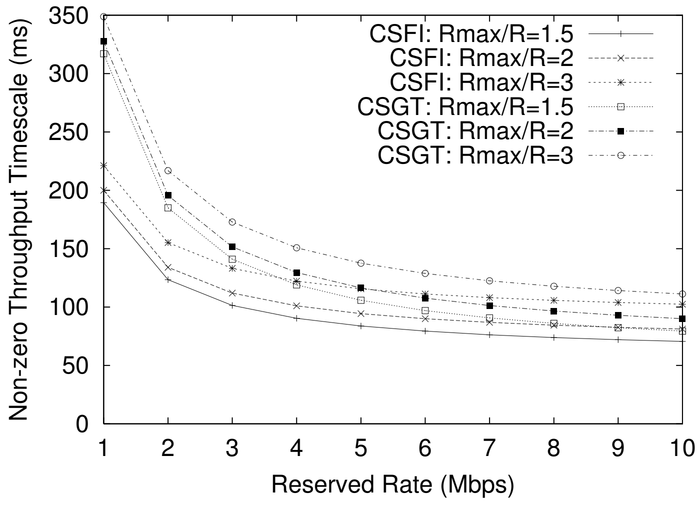


Figure 6.3: Application Throughput Guarantee of a CSGF Network

$$\le a * R + \sum_{f \in F} r_f\left(\frac{l}{r_m} + \frac{l}{r_f}\right)$$
$$\le a * R + \frac{l}{r_m}R + \sum_{f \in F} l$$

This implies:

$$\begin{aligned} W_m^{app}(t_1, t_2) \quad \ge \quad & r_m(t_2 - t_1) - r_m\left((H+1)\frac{l}{R} + \sum_{j=1}^{H-1}\pi_j + \sum_{j=1}^{H}\beta_{f,j} + D^{max}\right) \\ & -\frac{r_m}{R}W * l - \frac{r_m}{R}\sum_{f \in F} l - l \qquad (6.6) \end{aligned}$$

A CSGF network, therefore, provides a per-flow application throughput guarantee characterized by (6.6). To compare (6.6) to the application throughput guarantee of a CSGT network, we compute $\gamma_{f,H}^{net}$—the minimum timescale at which non-zero application throughput is guarantee—for an example network topology where link capacities are $100Mbps$, end-to-end propagation latency is $10ms$, $D^{max} = 10ms$, and path length is 10. Figure 6.3 plots $\gamma_{f,H}^{net}$ as a function of the reserved rate for a flow, for different values of R^{max}, the maximum rate that sources are allowed to achieve. We find that for flows with small reserved rates, the per-flow application throughput guarantee of a CSGF network may be even better than that of a CSGT network. This is because $\gamma_{f,H}^{net}$ is inversely proportional to r_f; the throughput guarantee provided to the aggregate in a CSGF network is, therefore, better than that provided to a micro-flow in a CSGT network. Since the unfairness measure on application throughput in a CSGF network is small, the throughput guarantee provided to a micro-flow in a CSGF network may be better than in a CSGT network. The difference between the two networks is less for flows with large bit-rates. More importantly, we find that a CSGF network is capable of providing application throughput guarantees at small time-scales of hundreds of milliseconds.

Network Throughput Guarantee Since the unfairness measure that characterizes fairness in network throughput for a CSGF network is large, $\gamma_{f,H}^{net}$ —the minimum time-scale at which non-zero network throughput is guaranteed to a flow f— in the corresponding throughput guarantee of a CSGF network is also large (of the order of seconds). A CSGF network, therefore, does not provide per-flow network throughput guarantees comparable to those provided by CSGT or core-stateful networks.

6.4 Summary

In this chapter, we present the Core-stateless Guaranteed Fair (CSGF) network architecture—the *first* work-conserving core-stateless architecture that provides end-to-end fairness guarantees. We develop the architecture in two steps. First, we show that for a network to provide fairness guarantees, it must also provide throughput guarantees. Second, we demonstrate that a set of two mechanisms—fair access at the edge and aggregation of micro-flows in the core—when used in conjunction with a CSGT network, lead to the design of a CSGF network architecture that provides deterministic fairness guarantees. We find that the fairness guarantees provided by a CSGF network on application throughput are better than those provided by core-stateful networks, although fairness guarantees on network throughput are worse. Similarly, application throughput guarantees of a CSGF network are comparable to those provided by core-stateful networks, whereas network throughput guarantees are not. As part of future work, we plan to investigate the design of core-stateless networks that provide better fairness guarantees on network throughput as well.

Chapter 7

Scalability Evaluation on a Programmable Router Platform

We have shown that our core-stateless architectures come close to their core-stateful counterparts in providing per-flow guarantees. We next address the question: *what are the scalability limits of routers in the core-stateless network architecture?* The performance of routers is challenged because they operate under stringent *time* and *space* constraints.

1. The time constraints arise from the need to operate the router at "line speeds"; in particular, the amount of time spent in processing packets in a router to realize a service must be kept within a time budget (see Table 1.1). Routers in different network architectures perform different sets of functions, with varying levels of complexity; the packet processing speeds of routers in different architectures may, therefore, differ.

2. The space constraints arise because routers need to maintain and update complex data structures (for instance, per-flow state or routing state). These data structures often occupy a large amount of the available high-speed memory;

if a router is to provide simultaneously multiple functionalities, the limited amount of high-speed memory available in these routers becomes a constraint. Further, the overhead of accessing (often with mutual exclusion) these data structures can prevent the router from operating at line speeds.

We evaluate the performance of routers in different network architectures by comparing (1) the link speeds to which they can scale, and (2) the high-speed memory requirements they impose. The specific values taken by these two quantities depends not only on the network architecture, but also on the router platform. In this chapter, therefore, we evaluate the *relative* performance of different network architectures on the same router platform. The natural question to address is: *what platform should be used?* We address this question in Section 7.1.

For our scalability evaluation, we examine the core functions needed in routers in each architecture (Section 7.2), and evaluate these building blocks along the two dimensions of time and space in Sections 7.3 and 7.4, respectively. In Section 7.5, we then evaluate routers, constructed using one or more of these building blocks, in different network architectures.

7.1 Choice of Implementation Platform

Traditionally, high performance routers have been designed using single-function, Application Specific Integrated Circuits (ASICs) that process packets at high speeds. Generally, these ASICs have long design and development times and are difficult to upgrade to add new functionality. To allow rapid deployment of new network services, next-generation routers are likely to be built using programmable network processors that support the programming of packet processing functions in software. For our implementation, therefore, we use Intel's IXP1200 programmable router platform [32]. In what follows, we first describe briefly the architecture[1] of the IXP1200 network processor, and introduce the constraints and challenges in programming network services on it.

The IXP1200 Router Platform

Hardware Overview Intel's IXP1200 network processor platform contains 4 MAC ports, a StrongARM core processor, six RISC CPUs (known as microengines), a proprietary bus (the 64-bit 66MHz IX bus) controller, a PCI controller, control units for accessing off-chip SRAM and DRAM memory, and a small amount (4KB) of on-chip scratchpad memory (see Figure 7.1). The StrongARM Core is used for control path processing, such as handling slow path exception packets, managing routing tables, and other network state information. Microengines, on the other hand, are used for data path processing; they process multiple packets in parallel. Each microengine is associate with a 4KB instruction store. Both the StrongARM and the microengines are clocked at 200MHz.

To enable a network processor to process packets at line speeds, it is essential to hide the latency incurred while accessing memory during packet processing. To achieve this, each microengine supports 4 hardware threads; a microengine can switch context from one hardware thread to another in a single cycle. Lock acquisition is implemented on the IXP1200 using a single queue to which the request for any lock—whether available or not—is enqueued. Lock-requests are served in the FIFO order from this queue, which may lead to delays due to head-of-line blocking.

Although not explicitly required, the most natural use of DRAM is to buffer packets. This is a function of the size (256MB for our evaluation system) and the speed of memory access ($33 - 40$ cycles). SRAM is the natural place to store frequently accessed control information, such as routing table, per-flow state, etc.

[1]Our overview of the IXP1200 architecture is sufficient to understand the implementation challenges; for a detailed description of IXP1200, see [32].

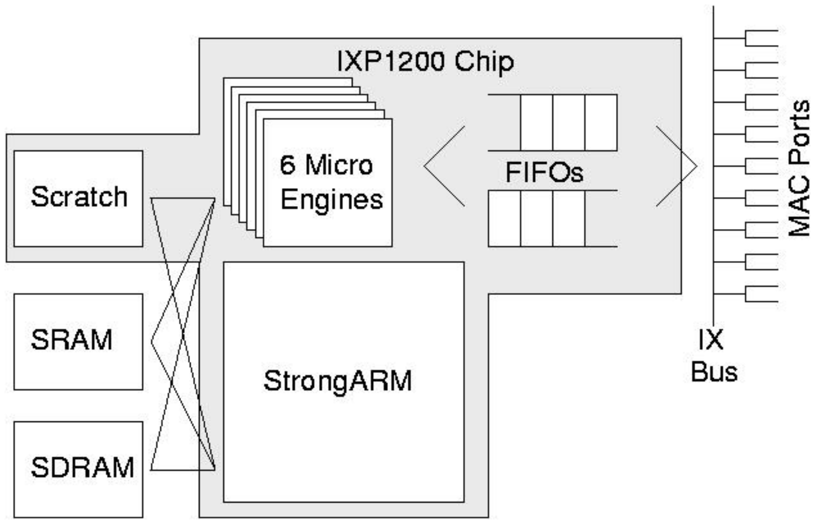


Figure 7.1: Block Diagram of the IXP1200 System

SRAM is relatively small in size ($8MB$ in our case) and has a much smaller access time ($16-20$ cycles). The on-chip scratchpad is used to read and write short control messages and data that are shared between microengines and the StrongARM.

IXP1200 Software Architecture Packet processing on the I/O-optimized IXP1200 router platform is divided into two stages: the *receive* stage and the *transmit* stage. Each thread is statically assigned to a port—threads assigned to input ports execute the receive stage and threads assigned to output ports execute the transmit stage. A receive thread transfers an incoming packet to the DRAM. It processes the packet, identifies the outgoing port (either through the per-flow state or the routing table), and enqueues the packet descriptor in the queue (may be a priority queue) corresponding to the outgoing port. When an outgoing port becomes available for transmission, the corresponding transmit thread dequeues a packet descriptor from the queue for that port, and transfers the corresponding packet from the DRAM to the port.

Due to the limited amount of instruction cache available per micro-engine, it is desirable to execute the same code on all threads of a given micro-engine. In addition, when different threads within the same micro-engine execute similar code, their memory access patterns are similar, and accesses can be interleaved efficiently. Memory latencies can, therefore, be hidden effectively. It follows that each one of the 6 micro-engines can either be a *receive* or a *transmit* micro-engine, with *all* 4 threads on it executing either the receive or the transmit stage, respectively. The 4 threads on each receive (or transmit) micro-engine are statically assigned to the 4 input (or output) ports.

7.2 Router Building Blocks in Different Network Architectures

Consider the core functions needed in routers in the following network architectures:

1. *FIFO Networks*

   The main function performed by routers in conventional FIFO IP networks is that of routing. IP routers maintain per-destination routing state, and for every incoming packet, search the state to identify the outgoing port on which to forward the packet. Packets destined for the same outgoing port are transmitted in the FIFO order.

   A simpler conceivable FIFO network is one that uses *source-routing*, instead of maintaining and using routing state at all routers. Routing state encoded in the packet header at the edge of the network is used to identify the output port. We use routers in such a network as the base-case against which to compare the performance of routers in more complex network architectures.

2. *Integrated Services (IntServ) Network Architecture*

   An IntServ network requires *all* routers to maintain per-flow state and perform packet classification to identify the flow to which an incoming packet belongs. For traffic classes that require flow identification, routing state can be maintained as part of the flow state; this eliminates the need to perform IP routing for every incoming packet. However, in *multi-class* networks, some classes—for instance, the best-effort traffic class—do not require flow identification; to support such classes, the router needs to maintain routing state.

Additionally, each router employs a scheduling algorithm that assigns priority values to packets and maintains a priority queue to transmit them in the increasing order of their priorities.

3. *Core-stateless Network Architecture*

   In a core-stateless architecture, such as the one proposed in this dissertation, per-flow classification and state maintenance is performed only at the *edge* routers of the network. Core routers use flow state encoded in the packet headers by the edge routers.

   The core routers maintain and use routing state in order to forward packets. However, if source-routing is used, routing state can also be encoded in the packet headers for this traffic class.

   Each router also maintains a priority queue to transmit packets in the increasing order of their priorities.

4. *MPLS-based Guaranteed Services Architecture*

   An MPLS network [51] performs flow classification only at the edge of the network. Core routers do maintain per-flow state, but use a flow-identifier encoded in the packet header to directly index into the flow state. Routing state can be included in the per-flow state; in multi-class networks, however, routers may need to maintain routing state to support service classes that do not require flow identification.

Each router also maintains a priority queue to transmit packets in the increasing order of their priorities.

To summarize, routers in the above architectures need to perform one or more of the following main functions: (1) flow classification and per-flow state maintenance; (2) routing and per-destination state maintenance; and (3) packet ordering. We examine each of these functions in some detail below.

7.2.1 Routing

Routing refers to the process of identifying the next-hop for a packet, based on the destination IP address. Since IP addresses are hierarchically allocated, routing tables at routers generally maintain next-hop information for IP address prefixes (that represent a collection of hosts with the same IP address prefix). Route selection for an incoming packet at a router, therefore, involves determining the *longest prefix-match* in the routing table for the destination host IP address. Several Trie-based schemes [9, 56] proposed in the literature are well-suited for this function.

7.2.2 Flow Classification

Flow classification is the process of locating the descriptor of the flow to which an incoming packet belongs. For the purpose of this evaluation, we define a *flow to be a sequence of packets flowing between two specific application end-points.* In case of IP networks, these two points are completely specified by the 5-tuple each packet carries; the elements of the 5-tuple are the IP addresses of the source and the destination, the transport protocol ID, and the port numbers at the source and the destination. Hence, the flow classification problem reduces to an exact match on the 5-tuple.

We consider the use of *chain-hashing* for solving the exact match on 5-tuples. Three reasons motivate the selection of hashing as a flow identification mechanism.

First, the basic hashing algorithm is simple and can be implemented efficiently in routers. Second, hashing schemes have a very good average-case performance. Finally, hash-based classification schemes can be analyzed for their worst case behavior [29], and hence are suitable to engineering.

7.2.3 Sorting

Link scheduling algorithms that enable routers to provide per-flow guarantees, assign priority values to packets and require routers to transmit packets in the increasing order of their priorities. This implies maintaining a priority queue data structure for each outgoing port, into which incoming packets are enqueued, and from which packets are dequeued for transmission. Priority queue implementation with logarithmic complexity of the enqueue and dequeue operations are well-known [16]. Recently, though, *constant-time* implementations of priority queues, that trade sorting accuracy for processing complexity, have been proposed [41, 49]. These implementations use variants of the Bin-sort [8] sorting algorithm, where each bin represents a range of priority values. We use such a constant-time priority-queue implementation; we refer to it as *Priority Bins.* Incoming packets are enqueued into the bin corresponding to their priority value. A dequeue operation involves searching through a *bit-vector* that indicates whether each bin is empty or not—packets are dequeued for transmission from the first non-empty bin. It can be shown that the approximate sorting of packets in Priority Bins increases the upper bound on the delay suffered by packets in the router queue, only by *binSize*, the length of the time interval represented by a bin.

Observe that *Priority Bins* can guarantee constant-time operations only if the number of bins is bounded. This implies that that the range of priority values assigned to packets simultaneously present in the queue, should be bounded. In Appendix C, we derive the following bounds on the number of packets ($maxP$) that may be present at a router, as well as the range of their deadline values (*maxRange*), assuming that the summation of the peak rates at which sources transmit does not exceed the link capacity:

$$\begin{aligned} maxRange &\simeq \sum_{j=1}^{h_f-1}(\pi_j - \widehat{\pi}_j) + (h_f - 1) * \frac{l_f}{r_f} + 2\beta_{h_f} \\ maxBytes &\simeq C_{h_f} * \sum_{j=1}^{h_f-1}(\pi_j - \widehat{\pi}_j) + \sum_{f \in F}(h_f - 1) * l_f \end{aligned}$$

where h_f is the hop-count of the router on the end-to-end path of flow f. The maximum number of packets, $maxP$, can be calculated as: $maxP = \frac{maxBytes}{l^{min}}$, where l^{min} is the minimum packet size. For the example network we considered before, the maximum number of flows in a priority queue for a $100Mbps$ link is around 1000. Using this we get $maxP \simeq 10000$. Since there are 4 ports on an IXP router, the maximum number of flows and packets would be a total of around 4000 and 40000 respectively. We use these computations in our scalability evaluation of Priority Bins.

In what follows, we first evaluate each of the above building blocks along the *time* and *space* dimensions, and then evaluate routers in different network architectures constructed using one or more of these building blocks.

7.3 Evaluation Along the *Time* Dimension

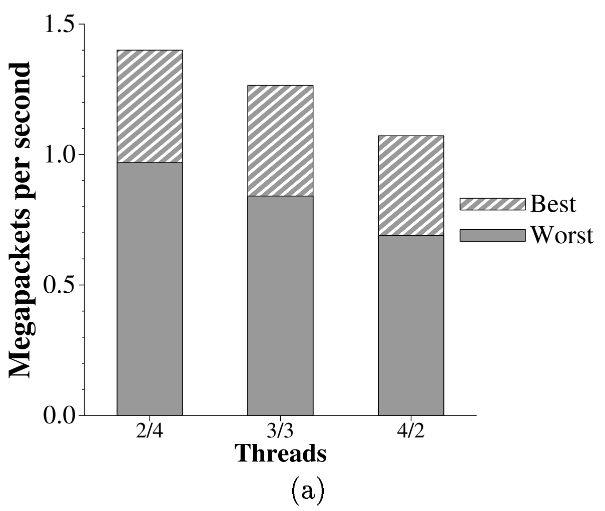
Our aim is to compare, through experiments, the throughput of hash-based flow classification, routing, and packet ordering, on the IXP1200 platform. For each of these components, we measure—in terms of packet processing rate (in millions of packets per second)—the *best-case* and *worst-case* performance, when different amounts of processing resources (e.g., microengines and threads) are allocated to the component. These measurements help us understand the *relative* cost of implementing these components on network processors.

Experimental Settings We conduct experiments with three different combinations of *(number of receive micro-engines, number of transmit micro-engines)* : $(2, 4)$, $(3, 3)$, and $(4, 2)$. All experiments are run in the IXP simulator environment shipped along with the hardware platform.

For experiments in this section, we consider an IXP router operating in a network with a maximum hop-length of 10, maximum end-to-end propagation latency of $10ms$, minimum reserved rate of $100Kbps$ for any flow, and minimum and maximum packet sizes of $64B$ (IXP *mpackets*) and $1500B$ ($\sim$ Ethernet maximum segment size), respectively. For such a network, using the formulation in Section 7.2.3, we can calculate: $maxP \simeq 16K$, and $nBins \simeq 1K$ (using $binSize \simeq 1ms$).

Priority Queue

We measure the throughput of the packet ordering component by measuring the time taken to perform a total of $30K$ enqueue and dequeue operations in a Priority Bins data structure. For Priority Bins, best-case performance is measured by ensuring that packets arrive at times close to their deadlines. This is because, the dequeue operation does not require multiple reads of the bit-vector in order to identify the first non-empty bin. The worst-case performance is measured by ensuring that packets arrive much ahead of their deadlines—the dequeue operation then requires several reads of the bit-vector (and hence several memory references)—and that the ensue and dequeue threads pay lock synchronization overheads by accessing the same bins simultaneously. Figure 7.2(a) plots the best-case and worst-case throughput for different combinations of the receive and transmit micro-engines. We observe that the effective throughput is best with a combination of $(2, 4)$ micro-engines, indicating that dequeue operations are, in general, more expensive than enqueue operations. The Priority Bins data structure is capable of supporting a throughput of $0.97Mpps$ in the worst-case, and $1.4Mpps$ in the best case.



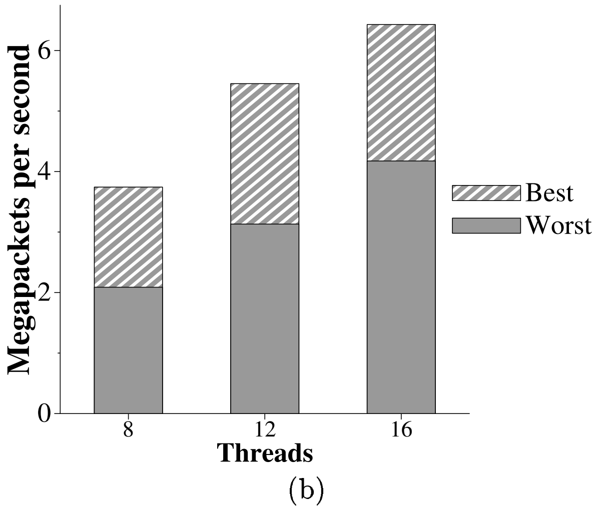


Figure 7.2: Throughput of (a) Priority Bins, and (b) Routing

Flow Classification

The processing complexity of chain-hashing depends on the number of memory references required to resolve hash collisions—we conduct experiments for different settings of the number of such memory references. Hashing is usually accompanied by modification of the flow descriptor hashed into. This requires the descriptor to be locked, and multiple threads hashing onto the same flow pay lock synchronization overheads. We conduct experiments with the best-case and worst-case scenarios, where different threads either hash onto different or the same flows, respectively. Figure 7.3 plots the throughput of chain-hashing, for these two scenarios, as a function of k, the number of hash collisions encountered. Note that the number of threads performing chain-hashing with the stage combinations of $(2, 4)$, $(3, 3)$, and $(4, 2)$, are 8, 12, and 16, respectively. We observe:

1. The throughput of the hash table decreases as k, the number of collisions encountered, increases. In fact, the memory reference overhead imposed by a large k overshadows the lock synchronization overheads incurred when dif-

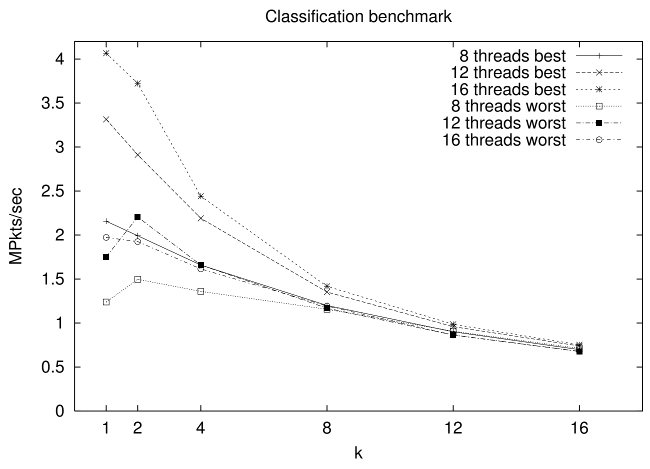


Figure 7.3: Throughput of Hashing

ferent threads access the same flow descriptor. The best-case and worst-case throughput are, therefore, similar for large values of k.

2. The best-case throughput increases almost linearly with increase in number of threads. The worst-case throughput, however, may decrease with increase in number of threads, due to lock synchronization overheads.

   Note, however, that multiple threads hash onto the same flow descriptor simultaneously only when multiple packets of the *same* flow arrive back-to-back. For high-speed links, this may not occur frequently; the average case lookup may, therefore, not suffer from as large overheads due to lock synchronization.

We observe that chain-hashing can support a throughput comparable to that of Priority Bins—for each of the stage combinations of $(2, 4)$, $(3, 3)$, and $(4, 2)$—if the maximum value of k is around 8.

Packet Routing

The processing complexity of a trie-based routing scheme depends on the number of trie levels traversed for each lookup. The worst-case lookup for a 4-stride trie

Component	*Throughput (Mpps)*					
	$(2,4)$		$(3,3)$		$(4,2)$	
	Best	*Worst*	*Best*	*Worst*	*Best*	*Worst*
Packet Ordering	1.40	0.97	1.27	0.84	1.07	0.69
Packet Classification	1.20	1.16	1.35	1.17	1.41	1.19
Routing	3.74	2.09	5.45	3.13	6.43	4.18

Table 7.1: Throughput of Different Components

on a 32-bit address space involves traversing 8 levels [53]. In the best-case, the lookup traverses only 2 levels. Figure 7.2(b) plots the best-case and worst-case trie throughput for different number of receive threads (not micro-engines) performing lookups. Note that route lookup requires no locking; the throughput, therefore, increases linearly with increase in number of threads.

Our experiments in this section indicate that the packet ordering and packet classification components are capable of sustaining similar throughput. The routing component, however, can sustain throughput multiple times that sustained by either of these components. Table 7.1 summarizes the results of this section.

7.4 Evaluation Along the *Space* Dimension

Our objective, in this section, is to compute the amount of memory space required by each of the building blocks in a typical core router. For the purpose of this computation, we do not restrict our attention to the IXP1200 platform available to us. Instead, we consider an example network based on state-of-the-art router configurations available [2].

Example Network Configuration We consider a network in which core routers operate on OC-192 links ($9.6Gbps$), the end-to-end propagation latencies are $10ms$, the maximum hop length is 10, and the minimum and maximum packet size are $64B$ and $1500B$ respectively. We assume that the minimum rate reserved by the network for any flow is $100Kbps$, which is close to the bandwidth requirements for high-fidelity audio. Therefore, the maximum number of flows, $maxF$, that could simultaneously traverse a core router link is around $100,000$. Using the analysis presented in Section 7.2.3. the maximum number of packets, $maxP$, that could be simultaneously present at a router queue—assuming that the sum of peak source rates does not exceed the link capacity—can be computed to be around $1,000,000$. The maximum range of deadlines carried by packets present simultaneously at a router queue is given by $maxRange = 1210ms$.

Sorting

For each packet in the SDRAM, a priority queue implementation maintains a descriptor in the SRAM. The Priority Bins algorithm, in addition to such $maxP$ packet descriptors, allocates space for the bins—2 words per bin for the *head* and *tail* pointers of the FIFO queue, and 1 bit per bin for the *bit-vector*. The total number of bins to be provisioned is given by:

$$nBins = \frac{maxRange}{binSize}$$

where $binSize$ is the length of the time interval represented by each bin. On the IXP platform, the size of the array representing the bins has to be a power of 2, given by: $2^{\lceil \log_2 (nBins) \rceil}$.

As mentioned earlier, using Priority Bins increases the delay bound of a packet by $binSize$, that is, the value of the constant $\beta_{f,j}$ increases by $binSize$. The end-to-end network delay guarantee of a packet, therefore, increases by $H * binSize$, where H is the number of hops traversed by the packet. A larger value of $binSize$ however, reduces $nBins$, and therefore, the amount of memory space number of bins to be provisioned. The parameter $binSize$, therefore, provides a knob that controls the tradeoff between memory space requirement at a router and the tightness of the end-to-end guarantee. Figure 7.4 plots the space requirement against the bin-size,

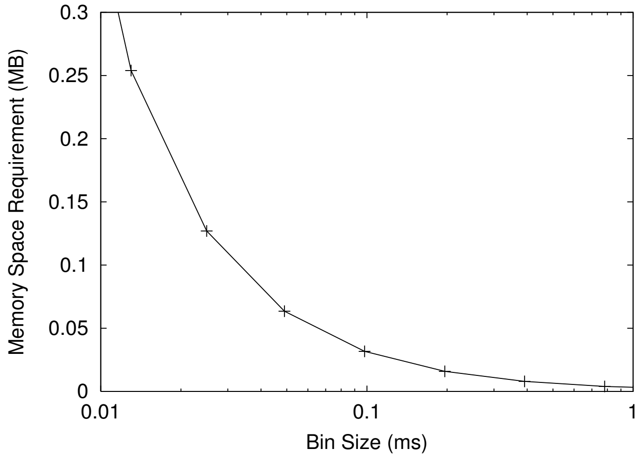


Figure 7.4: Priority Bins: Space vs. Accuracy tradeoff

for the network example described above. The bin-size, or the increase in per-node delay bound, is varied from 1% to 100% of the average link propagation latency of $1ms$. When $binSize$ is set to $0.1ms$, for instance, the memory space required for the Priority Bins data structure is around $32KB$.

Classification

When hashing is used to perform packet classification, in addition to the $nPorts * maxF$ flow descriptors, where $nPorts$ is the number of router input ports, a hash table needs to be maintained in memory. Let N denote the number of slots in the hash table. Then the hash table occupies $4N$ bytes of memory. Recall that packet classification can sustain throughput rates similar to packet ordering when k, the maximum number of collisions per hash table slot are within 8. A new flow may, therefore, be rejected by a router, if the slot to which it gets hashed already has k flow descriptors in its chain. Observe that the larger the value of N, the smaller the probability of a new flow finding k entries already present in a slot. In fact, it can be shown that B_{reject}, the probability of a flow being rejected by the packet classification stage, is given by [29]:

$$B_{reject} \quad = \quad Er\left(\rho/N, k\right) \tag{7.1}$$

where ρ is the average number of simultaneously-active flows at the router, and Er denotes the Erlang-B formula given by: $Er(\nu, c) = \frac{\nu^c/c!}{\sum_{i=0}^{c}(\nu^i/i!)}$. To ensure that the blocking probability is below a desired value, (7.1) can be used to compute the size of the hash table that should be provisioned. For instance, with $k = 4$ and $\rho = nPorts * maxF$, and $B_{reject} = 0.1\%$, the hash table occupies around $3.5MB$ of memory.

Packet Routing

The amount of memory space used by the trie routing structure depends on (1) the number of prefixes stored in the trie, and (2) the length of the prefixes. Current measurements estimate the number of prefixes in core routers to be around $100,000$ [45]. The average height of the path to a route entry in a 4-stride trie is around 2.76 [53]. Using these parameters, the memory space occupied by the trie on an average can be estimated as: $2^4 * 100,000 * 2.76$ words, which is around $17.6MB$. In the worst-case, however, the height of the paths to *all* route entries could be 8, which would make the memory usage to be around $51.2MB$.

7.5 Evaluation of Different Network Architectures

We now build upon the evaluation of router building blocks in Sections 7.3 and 7.4 to evaluate, along the time and space dimensions, the performance of core routers in different network architectures.

7.5.1 Processing Time Considerations

We implement routers for each of the network architectures described in Section 7.2. We conduct experiments with two sets of inputs: one for the best-case performance of each of the individual components (as described in the experiments in Section 7.3), and one for the worst-case performance of each. Input traffic arrives on all 4 input ports at the maximum rate supported by the receive stage, and is distributed equally across the 4 output ports. Figures 7.5-7.7 plot the cumulative throughput of the router, observed over all the output ports. We find that:

1. The router throughput is consistently higher with both the $(3, 3)$ and $(4, 2)$ stage combinations, than with the $(2, 4)$ combination. This is due to the fact that the receive stage is more expensive than the transmit stage in all the architectures we consider. For similar reasons, the best-case throughput observed with a $(4, 2)$ combination is slightly better than what is observed with a $(3, 3)$ combination. However, the worst-case throughput with a $(4, 2)$ combination may be significantly worse than the worst-case throughput observed with a $(3, 3)$ combination. This is due to the lock synchronization overheads paid when a large number of receive threads attempt to perform simultaneous dequeue operations on the Priority Bins. We conclude that the $(3, 3)$ combination is best for routers in the architectures we have considered.

2. Core routers in a core-stateless architecture, that employs conventional per-node IP routing, can operate within 12% and 26%, in the best-case and worst-case respectively, of the link speeds at which core routers in conventional FIFO networks can operate. The difference in throughput can be attributed mainly to the packet ordering component.

   Similar figures are observed when core-stateless architectures that employ source routing are compared to FIFO networks that employ source rout-

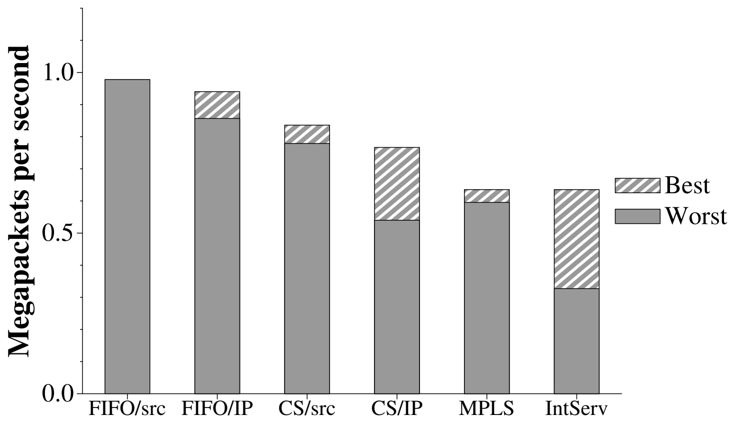


Figure 7.5: Router Throughput in Different Network Architectures: $(2, 4)$

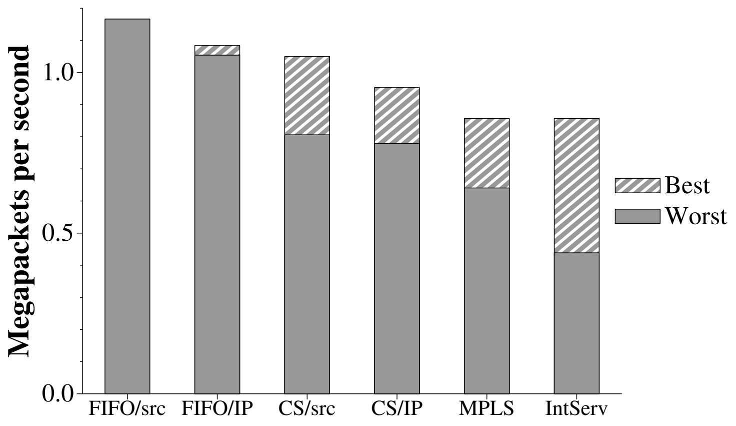


Figure 7.6: Router Throughput in Different Network Architectures: $(3, 3)$

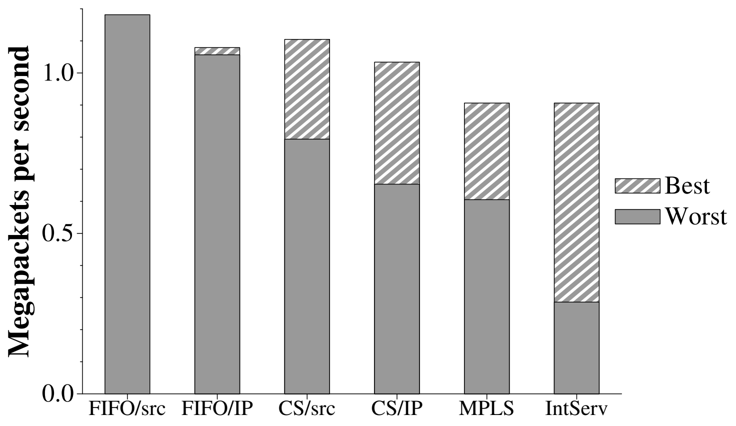


Figure 7.7: Router Throughput in Different Network Architectures: $(4, 2)$

ing. Routers in the former architecture can operate within 10% and 31%, in the best-case and worst-case respectively, of those in the latter architecture. Again, the difference can be attributed mainly to the packet ordering operation.

    In comparison to conventional FIFO IP routers, core routers in core-stateless networks that employ source routing operate within 1% and 26%, in the best-case and worst-case respectively.

3. Routers in the IntServ architecture can operate within 21% and 58%, in the best-case and worst-case respectively, of those in conventional FIFO IP networks. The greater difference in throughput, as compared to core-stateless networks, can be attributed mainly to per-flow classification and state maintenance.

4. The performance of routers in MPLS-based core-stateful networks lies within 21% and 39%, in the best-case and worst-case respectively, of conventional IP routers. The worst-case performance is better than in an IntServ network due to the use of simple tag-based classification mechanisms. Note, however, that routers still need to manipulate per-flow state and pay the associated locking overheads; the throughput of these routers is, therefore, lower than in core-stateless networks.

Our experiments indicate that core-stateless architectures that employ source routing come closest to meeting the throughput performance of conventional IP routers. Table 7.2 lists the throughput for the different architectures, normalized with respect to the highest throughput—that of routers in FIFO networks that employ source routing.

7.5.2 Space Requirements

For the discussion in this section, we reconsider the example network described in Section 7.4, where $maxP = 1,000,000$, $maxF = 100,000$, and $maxRange = 1210ms$. We configure $binSize = 0.1ms$, which is 10% of the single-link propagation latency, which implies that Priority Bins add a total of just $1ms$ overhead to the end-to-end propagation latency of around $10ms$. Using this, the total number of bins to be provisioned is calculated as $nBins = 16K$ (with $binSize = 1ms$, $nBins = 2K$). Table 7.2 summarizes the following discussion on the space requirements of routers in different architectures.

FIFO Networks To compute the memory space needed for packet descriptors maintained in the SRAM, observe that each descriptor maintains a pointer to the actual packet in the DRAM, the size of the packet, and a pointer to maintain the FIFO linked list. Since memory can be allocated only in powers of 2, the size of a packet descriptor is 4 words. For $maxP = 1,000,000$ packets, this adds up to around $16MB$ of SRAM occupied by the FIFO packet queue per router output port. In FIFO networks that employ conventional IP routing, a typical route table would additionally occupy around $18MB$ of SRAM (see Section 7.4).

IntServ Networks Flow descriptors in IntServ routers maintain a pointer to the routing information, the 5-tuple used for flow identification, the rate reserved for the flow, the deadline assigned to the last packet by the scheduling algorithm, and the pointer used to maintain the linked list in the chain hash-table. Computed as a power of 2, this adds up to 8 words per descriptor, which makes the total memory occupied by the $maxF$ flow descriptors to be around $3.1MB$, per router input port. In addition, the hash table occupies space of the order of $1MB$.

Packet descriptors occupy $16MB$ of memory as discussed above; the Priority Bins data-structure additionally occupies only around $32KB$. In multi-class

Architecture	*Relative Throughput*		*Fast-path Memory Required (MB)*	
	Best	***Worst***	***Single-class***	***Multi-class***
FIFO/source-routing	1.000	1.000	$nPorts * 16$	$nPorts * 16 + 18$
FIFO/IP	0.930	0.904	$nPorts * 16 + 18$	$nPorts * 16 + 18$
Core-stateless/source-routing	0.900	0.692	$nPorts * 16$	$nPorts * 16 + 18$
Core-stateless/IP	0.817	0.668	$nPorts * 16 + 18$	$nPorts * 16 + 18$
MPLS-based	0.735	0.549	$nPorts * 19$	$nPorts * 19 + 18$
IntServ	0.735	0.376	$nPorts * 20$	$nPorts * 20 + 18$

Table 7.2: Time and Space Complexities of Different Architectures

networks, the route table occupies around $18MB$.

Core-stateless Networks Packet descriptors and Priority Bins occupy around $16MB$ of memory. In multi-class networks, or in networks that use conventional IP routing, the route table occupies around $18MB$.

MPLS-based Guaranteed Services Networks The total memory occupied by the flow descriptors is around $6.1MB$, as in IntServ routers. The packet descriptors and bins occupy around $16MB$. In multi-class networks, the route table occupies around $18MB$.

The above discussion indicates that the fast-path memory requirements imposed on core routers in core-stateless networks are similar to those in conventional IP routers, whereas may be 18% to 25% larger in MPLS and IntServ networks, respectively. Further, the fast-path memory requirement of routers in MPLS and IntServ networks increases linearly with increase in the number of flows simultaneously traversing a router; since traffic demands are expected to increase with time [33], memory requirements in these architectures are expected to be higher than the above for future networks.

7.6 Summary

In this chapter, we evaluate the scalability limits of routers that implement our core-stateless architecture in comparison to those in conventional FIFO networks and IntServ networks. We use the approach of evaluating the core functions—namely, routing and per-destination state maintenance, per-flow classification and state maintenance, and packet ordering—needed by routers in these architectures. Our results indicate that:

1. Core routers in core-stateless networks that employ source routing can support best-case packet-processing speeds similar to those in conventional FIFO IP networks; they halve the gap between the worst-case packet-processing speeds of routers in FIFO and IntServ networks.

2. The memory space requirement of core routers in a core-stateless architecture is similar to that of routers in conventional FIFO networks.

Chapter 8

Conclusions

The advent of network applications with stringent timeliness requirements presents network architects with the opportunity of designing networks that provide per-flow service guarantees. In contrast, the rapid increase in link speeds and traffic demands presents the challenge of making such network architectures scalable and efficient. In this dissertation, we design network architectures that simultaneously achieve the above requirements of: (i) providing per-flow service guarantees, (ii) scaling to high-speed links and large number of flows, and (iii) utilizing resources efficiently.

8.1 Summary of Contributions

Existing network designs are either scalable (FIFO networks) or provide per-flow guarantees (Integrated Services, or IntServ, networks), but not both. To explore both ends of this spectrum of network designs, we address two questions: (1) *can preventing bursty traffic from entering the network enable FIFO networks to provide per-flow service guarantees*; and (2) *is it possible to provide end-to-end service guarantees, similar to those provided by IntServ networks, while removing per-flow computation from core routers?*

We address the first question in two steps: (1) we develop an analytical model that yields a closed-form characterization of the end-to-end performance of constant bit-rate (CBR) flows in FIFO networks under *asymptotic* conditions of network utilization and path length; and (2) we conduct simulations to verify the set of *non-asymptotic* and realistic conditions under which the results of the model continue to o hold. Our results indicate that CBR flows become heavy-tailed at moderate-to-large levels of utilization in large-scale networks. We conclude that FIFO scheduling is inadequate to design scalable networks that devote a significant fraction of available resources to support customers that require per-flow service guarantees.

To address the second question, we explore the design of core-stateless networks that provide per-flow service guarantees without maintaining per-flow state in the core routers. Our approach blends theory and practice. We address the theoretical aspects of designing core-stateless networks in two steps: (1) we understand the end-to-end service guarantees that can be provided in core-stateful networks; and (2) we design core-stateless networks that provide end-to-end service guarantees similar to core-stateful networks. On the practice front, we design and implement a router prototype to evaluate the feasibility of deploying our core-stateless architecture, and investigate the scalability of routers.

In understanding the end-to-end service guarantees provided by core-stateful networks, we conduct the *first* tight end-to-end fairness analysis of a network of fair queuing servers. We first argue that it is difficult to extend existing single-node fairness analysis to an end-to-end analysis of a network where each node may employ a different fair scheduling algorithm. We then present a two-step approach for end-to-end fairness analysis of heterogeneous networks. First, we define a class of scheduling algorithms, referred to as the *Fair Throughput* (FT) class, and prove that most known fair scheduling algorithms belong to this class. Second, we develop an analysis methodology for deriving the end-to-end fairness bounds for a network of FT servers. Our analysis is general and can be applied to heterogeneous networks where different nodes employ different scheduling algorithms from the FT class. We leverage past work to understand the end-to-end delay and throughput guarantees provided by core-stateful networks.

To derive core-stateless networks that provide delay guarantees, we design a methodology to transform any core-stateful network of Guaranteed Rate (GR) servers to a core-stateless version (CSGR) that provides the same end-to-end delay guarantee. Since the GR class is fairly general, this methodology provides a tool to design a wide range of core-stateless networks that provide delay guarantees. For instance, it is possible to design a core-stateless Delay-EDD network, that decouples the delay and rate guarantee.

Next, we propose the Core-stateless Guaranteed Throughput (CSGT) network architecture—the *first* work-conserving network architecture that provides throughput guarantees to individual flows over finite time-scales, but without maintaining per-flow state in core routers. We develop the architecture in two steps. First, we show that for a network to provide end-to-end throughput guarantees, it must also provide end-to-end delay guarantees. Second, we demonstrate that two mechanisms —tag re-use and source rate control— when integrated with a work-conserving, core-stateless network that provides end-to-end delay guarantees, lead to the design of CSGT network that provides end-to-end throughput bounds within an *additive constant* of what is attained by a core-stateful network of fair rate servers. We demonstrate that the constant is small for current network topologies.

As a final step in designing core-stateless networks, we propose the Core-stateless Guaranteed Fair (CSGF) network architecture—the *first* work-conserving core-stateless architecture that provides deterministic fairness guarantees. We develop the architecture in two steps. First, we show that for a network to provide fairness guarantees, it must also provide throughput guarantees. Second, we demonstrate that a set of two mechanisms—fair access at the edge and aggregation of flows in the core—when integrated with a CSGT network that provides throughput guarantees, lead to the design of CSGF networks that provide fairness guarantees. The fairness guarantees provided by a CSGF network on application throughput are comparable to those provided by core-stateful networks.

Our core-stateless architectures come close to their core-stateful counterparts in providing per-flow guarantees. We next evaluate the scalability of routers in our architectures—and compare them to routers in FIFO and IntServ networks—by implementing them on a programmable router platform. Our results indicate that core routers in a core-stateless architecture, coupled with a source-routing mechanism, can match closely the performance of routers in conventional IP networks.

Appendix A

Applying Approach of [59] to Virtual Clock

We apply the approach adopted in [59] for Jitter Virtual Clock to its work-conserving version, Virtual Clock, in this section and demonstrate that the delay guarantees of Virtual Clock are not preserved in its core-stateless version.

Let H be the total number of nodes that packets of flow f traverse. Let K be the total number of flow f packets transmitted. If we introduce a per-packet *slack variable*, δ_f^k, such that,

$$\delta_f^k \quad \geq \quad VC_{f,j}^{k-1} - a_{f,j}^k, \qquad j = 2, ..., H; k = 2, ..., K \tag{A.1}$$

and we compute the new *core virtual clock* values as:

$$VCore_{f,j}^k \quad = \quad a_{f,j}^k + \delta_f^k + \frac{l_f^k}{r_f}, \qquad j = 2, ..., H; k = 1, ..., K \tag{A.2}$$

where $\delta_f^1 = 0$, then we can calculate $VCore_{f,j}^k$ purely on the o basis of variables that can be encoded in the packet by nodes $j \geq 1$.

Note that, from Equations (4.6), (A.1), and A.2, we get:

$$VCore_{f,j}^k \quad \geq \quad VC_{f,j}^k, \qquad \forall j, k, f \tag{A.3}$$

Using A.3, we can show that a server scheduling packets in increasing order of their *core virtual clock* values would transmit a packet by $(VCore^k_{f,j} + \beta_{f,j})$, where $\beta_{f,j} = \frac{l^{max}_j}{C_j}$ (the proof is similar to the one for Theorem 4). Hence, the following upper bound on $a^k_{f,j}$ exists:

$$a^k_{f,j} \leq VCore^k_{f,j-1} + \beta_{f,j-1} + \pi_{j-1} \tag{A.4}$$

For computing δ^k_f, note that $\delta^1_f = 0$.

Further, in order to ensure Inequality (A.1), we need to make sure that δ^k_f is no smaller than the largest value that its *RHS* can take. This is given by upper and lower bounds on $VC^{k-1}_{f,j}$ and $a^k_{f,j}$ respectively as follows:

$$\begin{aligned}
VC^{k-1}_{f,j} &= \max(a^{k-1}_{f,j}, VC^{k-2}_{f,j}) + \frac{l^{k-1}_f}{r_f} \\
&\leq a^{k-1}_{f,j} + \delta^{k-1}_f + \frac{l^{k-1}_f}{r_f} && (from(A.3)) \\
&\leq VCore^{k-1}_{f,j-1} + \pi_{j-1} + \beta_{f,j-1} + \delta^{k-1}_f + \frac{l^{k-1}_f}{r_f} && (from(A.4)) \\
&\leq a^{k-1}_{f,j-1} + 2(\delta^{k-1}_f + \frac{l^{k-1}_f}{r_f}) + (\pi_{j-1} + \beta_{f,j-1}) \\
&\;. \\
&\;. \\
&\leq a^{k-1}_{f,2} + (j-1)(\delta^{k-1}_f + \frac{l^{k-1}_f}{r_f}) + \sum_{i=2}^{j-1} (\pi_i + \beta_{f,i}) \\
&\leq VC^{k-1}_{f,1} + (j-1)(\delta^{k-1}_f + \frac{l^{k-1}_f}{r_f}) + \sum_{i=1}^{j-1} (\pi_i + \beta_{f,i})
\end{aligned}$$

$$\begin{aligned}
a^k_{f,j} &\geq a^k_{f,j-1} + \pi_{j-1} + \frac{l^k_f}{C_j} \\
&\geq a^k_{f,j-2} + \sum_{i=j-2}^{j-1} (\pi_i + \frac{l^k_f}{C_i}) \\
&\;. \\
&\;.
\end{aligned}$$

$$\geq \quad a_{f,1}^k + \sum_{i=1}^{j-1} (\pi_i + \frac{l_f^k}{C_i})$$

Therefore, at node j, the maximum possible value that δ_f^k would be required to exceed is given by:

$$\delta_f^k \quad \geq \quad (VC_{f,1}^{k-1} - a_{f,1}^k) + (j-1)(\delta_f^{k-1} + \frac{l_f^{k-1}}{r_f}) + \sum_{i=1}^{j-1} (\beta_{f,i} - \frac{l_f^k}{C_i})$$

Since $(\beta_{f,i} \geq l_f^k / C_i)$, the RHS is maximized for $j = H$, and therefore, the least sequence of δ_f^k that would satisfy Inequality (A.1) is given by:

$$\begin{aligned} \delta_f^1 &= 0 \\ \delta_f^k &= \max(0,\ (VC_{f,1}^{k-1} - a_{f,1}^k) + (H-1)(\delta_f^{k-1} + \frac{l_f^{k-1}}{r_f}) + \sum_{i=1}^{H-1} (\beta_{f,i} - \frac{l_f^k}{C_i}))(\text{A.5}) \end{aligned}$$

Delay Guarantee of a Network of Core-Stateless Virtual Clock Servers

Consider a flow f traveling through a network of H Core-Stateless Virtual Clock Servers. Let $d_{f,j}^k$ be the time at which packet p_f^k gets transmitted at the j^{th} node. Then, the end-to-end delay guarantee of the k^{th} packet is computed as:

$$\begin{aligned} d_{f,H}^k - VC_{f,1}^k &\leq VCore_{f,H}^k + \beta_H - VC_{f,1}^k \\ &\leq a_{f,H}^k - VC_{f,1}^k + \delta_f^k + \frac{l_f^k}{r_f} + \beta_H \\ &\quad . \\ &\quad . \\ &\leq a_{f,1}^k - VC_{f,1}^k + H(\delta_f^k + \frac{l_f^k}{r_f}) + \sum_{i=1}^{H-1} (\pi_i + \beta_i) + \beta_H \\ &\leq H(\delta_f^k + \frac{l_f^k}{r_f}) + \sum_{i=1}^{H-1} (\pi_i + \beta_i) + \beta_H \end{aligned}$$

This delay guarantee for the k^{th} packet of a flow depends on the value of its slack variable, δ_f^k. To get a feel for the lower bound on δ_f^k, consider a well-behaved source, that sends its packets at a constant bit-rate, r_f, i.e.:

$$a_{f,1}^k = VC_{f,1}^{k-1}$$

Substituting in Equation (A.5), we get:

$$\begin{aligned}
\delta_f^1 &= 0 \\
\delta_f^2 &= (H-1) * \frac{l_f^1}{r_f} + \sum_{i=1}^{H-1} (\beta_{f,i} - \frac{l_f^k}{C_i}) \\
\delta_f^3 &= (H-1)^2 * \frac{l_f^1}{r_f} + (H-1) * \frac{l_f^2}{r_f} + H * \sum_{i=1}^{H-1} (\beta_{f,i} - \frac{l_f^k}{C_i}) \\
\delta_f^4 &= (H-1)^3 * \frac{l_f^1}{r_f} + (H-1)^2 * \frac{l_f^2}{r_f} + (H-1) * \frac{l_f^3}{r_f} + (H(H-1)+1) * \sum_{i=1}^{H-1} (\beta_{f,i} - \frac{l_f^k}{C_i}) \\
&. \\
&. \\
&.
\end{aligned}$$

We see that, as k grows, such a computation of the slack variable δ_f^k to eliminate per-flow state in core nodes fails to provide reasonable end-to-end delay guarantees for even constant bit-rate flows, and hence the approach used in [59] does not work for Virtual Clock.

Appendix B

Condition on W to Allow Large Throughput to be Achieved in a CSGT Network

Suppose $r' > r_f$ is the bottleneck bandwidth available to the flow f at time t_0 (and thereafter). Without loss of generality, assume that the first server is the bottleneck server[1]. We derive a condition on W such that if there is only a single packet at the bottlenecked first server (and in the network) at a time t_0, then there continues to be at least one packet at the bottlenecked server at all future times (given that the source has packets to transmit). If this is the case, then the bottleneck bandwidth available to flow f is not wasted.

Suppose packet p_f^1 arrives at the server queue at time t_0. Then it can be shown that packet p_f^k, (where packets $p_f^1, \ldots, p_f^k$ are transmitted back-to-back) would incur a maximum delay of $(\sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + (H+k-1) * l_f/r')$ before it departs from the network. The acknowledgment for packet p_f^k would reach the ingress node after an additional delay of at most D^{max}. The corresponding S_1 of this packet at

[1]Suppose this is not the case. Then even if the first server transmits packets once only every l_f/r' time units, the bottleneck server will remain backlogged.

the ingress node would be $S_1(p_f^k) = t_0 + (k-1)l_f/r_f$. Therefore, the tag-vector of packet p_f^k would definitely be available for re-use at the first server, if k satisfies: $\sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + (H+k-1) * l_f/r' + D^{max} \leq (k-1) * l_f/r_f$. This implies:

$$k \geq \frac{\sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + D^{max} + H * \frac{l_f}{r'}}{\frac{l_f}{r_f} - \frac{l_f}{r'}} + 1 \tag{B.1}$$

Now observe that, if the bottlenecked first server remains backlogged till the time the acknowledgment for p_f^k arrives, then the subsequent acknowledgments (spaced l_f/r' apart) clock the transmission of new packets, and the bottleneck server, that transmits a packet every l_f/r' units, would remain backlogged subsequently.

Further, due to source flow control, the tags for packet p_f^k would become available for re-use at most by the time $W + m_k$ packets arrive for transmission at the first server after t_0, where m_k is given by the term:

$$m_k = \left\lfloor \frac{\sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + (H+k-1)\frac{l_f}{r'} + D^{max}}{l_f/r_f} \right\rfloor \tag{B.2}$$

Therefore, if the time it takes for the first server to transmit $W + m_k$ packets at the rate r', is at least as large as the maximum time it takes for the acknowledgment of packet p_f^k to arrive after t_0, the first server always remains backlogged with packets to transmit. That is, the following condition would ensure that the server remains continuously backlogged: $(W + m_k) * \frac{l_f}{r'} \geq \sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + D^{max} + (H + k - 1) * \frac{l_f}{r'}$. From (B.1) and (B.2), this yields:

$$(W-1)l_f \geq r'(\sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + D^{max}) + H\ l_f \tag{B.3}$$

If, on the other hand, W does not satisfy condition (B.3) for the available bottleneck bandwidth r', then it can be seen, using a similar argument as above, that the throughput rate that the source can sustain, $R^{max}(W, r')$, is given by:

$$\simeq r_f + \frac{\left(1 - \frac{r_f}{r'}\right)(W-1) * l}{\sum_{j=1}^{H-1} \pi_j + \sum_{j=1}^{H} \beta_{f,j} + D^{max} + H * \frac{l_f}{r'}}$$

Appendix C

Computing Maximum Deadline Range and Queue Size

As representative of core-stateful and core-stateless networks that provide per-flow service guarantees, we consider those that employ scheduling algorithms from the GR [25] and CSGR [35] classes. Recall that in these frameworks, a deadline is computed for each incoming packet at a router, and the scheduling algorithm provides a *deadline guarantee*, that a packet would depart the router by its deadline (plus a constant).

Let $F_h(p_f^k)$ denote the deadline assigned to p_f^k, the k^{th} packet of flow f, at the h^{th} node in its path. The following relation exists between F_1 and F_h [25, 35]:

$$\begin{aligned} F_1(p_f^k) &= \max\left(F_1(p_f^{k-1}), a_{f,1}^k\right) + \frac{l_f}{r_f} \\ F_h(p_f^k) &\leq F_1(p_f^k) + \sum_{j=1}^{h-1} \pi_j + \sum_{j=1}^{h} \beta_j + (h-1) * \frac{l_f}{r_f} \end{aligned} \tag{C.1}$$

where l_f is the packet size for flow f, and $\beta_{f,j}$ is a constant that depends on the traffic and server characteristics at node j. If the sum of the reserved rate over all flows does not exceed the link capacity, algorithms in the GR and CSGR classes guarantee that p_f^k would depart node j by $F_j(p_f^k) + \beta_{f,j}$. If r_f,the reserved rate, is also the peak source rate, we get: $a_{f,1}^k \geq F_1(p_f^{k-1})$. Therefore, $F_1(p_f^k) = a_{f,1}^k + \frac{l_f}{r_f}$.

Maximum range of deadlines Consider a router. Let h_f denote the hop count of this router for packets of flow f. The maximum number of packets of flow f in the router queue can be computed by assuming that (1) every packet p_f^k that arrives into the network, zips through the first $h_f - 1$ hops, experiencing no queuing at these hops, and (2) every packet p_f^k experiences the maximum queuing delay at hop h_f, departing only at $F_{h_f}(p_f^k) + \beta_{h_f}$. Let $\widehat{\pi}_j$ denote the minimum propagation latency on the link connecting server j and $j+1$. Then, packet p_f^k can arrive at the h_f^{th} node as early as: $a_{f,1}^k + \sum_{j=1}^{h_f-1}(\widehat{\pi}_j + l_f/C_j)$. It follows that the largest deadline carried by any packet of flow f at node h_f at a given time t is given by: $F_{h_f}(p) = t + \sum_{j=1}^{h_f-1}(\pi_j - \widehat{\pi}_j) + \sum_{j=1}^{h_f-1}(\beta_j - l_f/C_j) + \beta_{h_f} + (h_f - 1) * \frac{l_f}{r_f}$. From the deadline guarantee of node h_f, the *smallest* deadline that a packet could carry at time t is $(t - \beta_{h_f})$. Assuming maximum-sized packets, and using a representative value of $\beta_{f,j} = l^{max}/C_j$, it follows that the maximum range of deadlines carried by packets of a flow f simultaneously present at a router queue is given by:

$$maxRange \simeq \sum_{j=1}^{h_f-1}(\pi_j - \widehat{\pi}_j) + (h_f - 1) * \frac{l_f}{r_f} + 2\beta_{h_f}$$

Consider an example network where the end-to-end path length of any flow is at most 10 hops, the end-to-end propagation latency is at most $10ms$, minimum and maximum packets sizes are $64B$ and $1500B$ respectively, and the minimum rate that a flow can reserve is $100Kbps$. The value of $maxRange$ for such a network is of the order of $100ms$.

Maximum queue size Note that the maximum number of bytes of flow f that can be present at a router queue simultaneously is no more than: $r_f * maxRange$. Let F denote the set of all flows traversing through the router. The following is an upper bound on the maximum number of bytes across *all* flows, present simultaneously at a router queue:

$$\begin{aligned} maxBytes &\simeq \sum_{f \in F} r_f * \left(\sum_{j=1}^{h_f - 1} (\pi_j - \widehat{\pi}_j) + (h_f - 1) * \frac{l_f}{r_f} \right) \\ &\leq C_{h_f} * \sum_{j=1}^{h_f - 1} (\pi_j - \widehat{\pi}_j) + \sum_{f \in F} (h_f - 1) * l_f \end{aligned}$$

Bibliography

[1] CSIM18 - The Simulation Engine. http://www.mesquite.com.

[2] Intel®IXP2800 Network Processor. http://www.intel.com/design/network/products/npfamily/ixp2800.htm.

[3] V. Agarwal, M.S. Hrishikesh, S.W. Keckler, and D.C. Burger. Clock Rate Versus IPC: The End of the Road for Conventional Microarchitectures. In *27th International Symposium on Computer Architecture (ISCA)*, June 2000.

[4] J.C.R. Bennett, K. Benson, A. Charny, W.F. Courtney, and J.Y.LeBoudec. Delay Jitter Bounds and Packet Scale Rate Gaurantee for Expedited Forwarding. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1502–1509, 2001.

[5] J.C.R. Bennett, K. Benson, A. Charny, W.F.Courtney, and J.Y. LeBoudec. Delay Jitter Bounds and Packet Scale Rate Guarantee for Expedited Forwarding. to appear in IEEE/ACM Transactions on Networking.

[6] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case Fair Weighted Fair Queuing. In *Proceedings of INFOCOM'96*, pages 120–127, March 1996.

[7] J.C.R. Bennett and H. Zhang. Hierarchical Packet Fair Queueing Algorithms. In *IEEE/ACM Transactions on Networking*, volume 5, pages 675–689, Oct 1997.

[8] R. Brown. Calendar Queues: A Fast $O(1)$ Priority Queue Implementation for the Simulation Event Set Problem. *Communications of the ACM*, 31(10):1220–1227, October 1988.

[9] Milind M. Buddhikot, Subhash Suri, and Marcel Waldvogel. Space Decomposition Techniques for Fast Layer-4 Switching. In *Protocols for High Speed Networks IV (Proceedings of PfHSN '99)*, pages 25–41, August 1999.

[10] Z. Cao, Z. Wang, and E. Zegura. Rainbow Fair Queueing: Fair Bandwidth Sharing Without Per-Flow State. In *Proceedings of IEEE INFOCOM*, March 2000.

[11] A. Charny, F. Baker, J. Bennett, K. Benson, J.-Y. LeBoudec, A. Chiu, W. Courtney, B. Davie, S. Davari, V. Firou, C. Kalmanek, K.K. Ramakrishnan, and D. Stiliadis. EF PHB Redefined. Nov 2000. Internet Draft.

[12] I. Chlamtac, A. Farago, H. Zhang, and A. Fumagalli. A Deterministic Approach to the End-to-end Analysis of Packet Flows in Connection-oriented Networks. In *IEEE/ACM Transactions on Networking*, volume 6, August 1998.

[13] D. Clark and W. Fang. Explicit Allocation of Best Effort Packet Delivery Service. *IEEE/ACM Transactions on Networking*, 1(6):362–373, August 1998.

[14] A. Clerget and W. Dabbous. TUF: Tag-based Unified Fairness. In *Proceedings of IEEE INFOCOM*, April 2001.

[15] K. Coffman and A. Odlyzko. The Size and Growth Rate of the Internet. March 2001. http://www.firstmoday.dk/issues/issue3_10/coffman/.

[16] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw Hill, 1996.

[17] R. L. Cruz. SCED+: Efficient Management of Quality of Service Guarantees. In *Proceedings of INFOCOM'98*, March 1998.

[18] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of ACM SIGCOMM*, pages 1–12, September 1989.

[19] A. DeSimone. Generating Burstiness in Networks: A Simulation Study of Correlation Effects in Networks of Queues. *ACM Computer Communication Review*, pages 24–31, 1991.

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

[21] L. Georgiadis, R. Guerin, V. Peris, and K.N. Sivarajan. Efficient Network QoS Provisioning Based on per Node Traffic Shaping. In *Proceedings of INFOCOM'96*, pages 102–110, March 1996.

[22] S.J. Golestani. A Self-Clocked Fair Queueing Scheme for High Speed Applications. In *Proceedings of INFOCOM'94*, 1994.

[23] P. Goyal. Packet Scheduling Algorithms for Integrated Services Networks. *PhD thesis, University of Texas at Austin, Austin, TX*, August 1997.

[24] P. Goyal, S.S. Lam, and H.M. Vin. Determining End-to-End Delay Bounds In Heterogeneous Networks. In *ACM/Springer-Verlag Multimedia Systems Journal*, 1996. Also appeared in the Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video, April 1995.

[25] P. Goyal and H.M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. In *IEEE/ACM Transactions on Networking*, volume 5, pages

561–571, August 1997. Also available as technical report TR95-30, Department of Computer Sciences, The University of Texas at Austin.

[26] P. Goyal, H.M. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM'96*, pages 157–168, August 1996.

[27] M. Grossglauser and S. Keshav. On CBR Service. In *Proceedings of INFOCOM'96*, pages 129–137, March 1996.

[28] R. Gruenenfelder. A Correlation Based End-to-End Cell Queueing Delay Characterization in an ATM Network. In *Proc. Thirteenth International Teletraffic Congress (ITC-13)*, volume 15, pages 59–64, June 1991.

[29] B. Hardekopf, T. Riche, J. Kaur, J. Mudigonda, M. Dahlin, and H. Vin. Scalability Analysis of Software-based Service-differentiating Routers Using Network Processors. *Technical Report, Department of Computer Sciences, University of Texas at Austin*, May 2001.

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

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

[32] E. Johnson and A. Kunze. *IXP1200 Programming.* Intel Press, 2002.

[33] P. Kaiser. A (R)evolutionary Technology Roadmap Beyond Today's OE Industry. *NSF Workshop on The Future Revolution in Optical Communications & Networking*, December 2000.

[34] D. D. Kandlur, K. G. Shin, and D. Ferrari. Real-Time Communication in

Multihop Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(10):1044–1056, October 1994.

[35] J. Kaur and H.M. Vin. Core-stateless Guaranteed Rate Scheduling Algorithms. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1484–1492, April 2001.

[36] J. Kaur and H.M. Vin. End-to-end Fairness Analysis of Fair Queuing Networks. In *Proceedings of the 23rd IEEE International Real-time Systems Symposium (RTSS)*, Dec 2002.

[37] L. Kleinrock. Queueing Systems, Volume 1: Theory. *John Wiley & Sons, New York, NY*, 1975.

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

[39] J.Y. LeBoudec and G. Hebuterne. Comments on "A deterministic approach to end-to-end analysis of packet flows in connection-oriented networks". *IEEE/ACM Transaction on Networking*, 6(24):422–431, 1998.

[40] C. Li and E. Knightly. Coordinated Network Scheduling: A Frameworkfor End-to-end Services. In *IEEE ICNP 2000)*, Nov 2000.

[41] J. Liebeherr and D.E. Wrege. Priority Queueing Schedulers with Approximate Sorting in Output Buffered Switches. In *IEEE Journal on Selected Areas in Communications*, volume 17, pages 1127–1145, June 1999.

[42] W. Matragi, C. Bisdikian, and K. Sohraby. Jitter Calculus in ATM Networks: Multiple Node Case. *IEEE INFOCOM '94*, June 1994.

[43] K. Nichols, V. Jacobson, and L. Zhang. An Approach to Service Allocation in the Internet. November 1997. Internet Draft.

[44] K. Nichols, V. Jacobson, and L. Zhang. A Two-bit Differentiated Services Architecture for the Internet. November 1997. ftp://ftp.ee.lbl.gov/papers/dsarch.pdf.

[45] University of Oregon Route Views Project. BGP Core Routing Table Size. 2002. http://www.antc.uoregon.edu/route-views/dynamics/.

[46] R. Pan, B. Prabhakar, and K. Psounis. CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *Proceedings of IEEE INFOCOM*, March 2000.

[47] A.K. Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1992.

[48] A. Privalov and K. Sohraby. Per-stream Jitter Analysis in CBR ATM Multiplexors. *IEEE/ACM Transaction on Networking*, 6(2), 1998.

[49] J. Rexford, A. Greenberg, and F. Bonomi. Hardware-efficient Fair Queueing Architectures for High-speed Networks. In *Proceedings of IEEE INFOCOM*, March 1996.

[50] J.W. Roberts and J.T. Virtamo. The Superposition of Periodic Cell Arrival Streams in an ATM Multiplexer. *IEEE Transactions on Communications*, 39(2):298–303, February 1991.

[51] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. January 2001. Internet RFC 3031.

[52] J. Sahni, P. Goyal, and H.M. Vin. Scheduling CBR Flows: FIFO or Per-Flow Queueing? In *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'99)*, pages 13–27, June 1999.

[53] M.A.R. Sanchez, E.W. Biersack, and W. Dabbous. Survey and Taxonomy of IP Address Lookup Algorithms. *IEEE Network*, 15(2):8–23, March 2001.

[54] S. Shenker and C. Partridge. Specification of Guaranteed Quality of Service. Available via anonymous ftp from ftp://ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-intserv-guaranteed-svc-03.txt, November 1995.

[55] K. Sohraby and A. Privalov. End-to-End Jitter Analysis in Networks of Periodic Flows. In *Proceedings of IEEE INFOCOM*, volume 2, pages 575–583, 1999.

[56] V. Srinivasan, George Varghese, Subhash Suri, and Marcel Waldvogel. Fast and Scalable Layer Four Switching. In *Proceedings of ACM SIGCOMM*, pages 191–202, September 1998.

[57] I. Stoica. Stateless Core: A Scalable Approach for Quality of Service in the Internet. *PhD thesis, Carnegie Mellon University, Pittsburgh, PA*, December 2000.

[58] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM'98*, Sept 1998.

[59] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow Management. In *Proceedings of ACM SIGCOMM'99*, Sept 1999.

[60] M. Venkatachalam, J. Kaur, and H.M. Vin. End-to-end Model for a Flow in the Internet. *Technical Report TR-01-32, Department of Computer Sciences, University of Texas at Austin*, August 2001.

[61] G.G. Xie and S.S. Lam. Delay Guarantee of Virtual Clock Server. *IEEE/ACM Transactions on Networking*, 3(6):683–689, December 1995.

[62] D. Yates, J.F. Kurose, D. Towsley, and M.G. Hluchyj. On per-session end-to-end delay distributions and the call admission problem for real-time applications with QOS requirements. In *Proceedings of ACM SIGCOMM*, pages 160–166, October 1993.

[63] H. Zhang. Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10), October 1995.

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

[65] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proceedings of ACM SIGCOMM'90*, pages 19–29, August 1990.

[66] Z.L. Zhang, Z. Duan, and Y.T. Hou. Virtual Time Reference System: A Unifying Scheduling Framework for Scalable Support of Guarantees Services. *IEEE Journal on Selected Areas in Communication, Special Issue on Internet QoS*, Dec 2000.

[67] Q. Zheng and K. Shin. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-switching Networks. *IEEE Transactions on Communications*, 42(3):1096–1105, March 1994.

Vita

Jasleen Kaur Sahni was born on July 1, 1974 in Dehradun, India, the daughter of Adarsh Kaur Sahni and Devinder Singh Sahni. She received the Bachelor of Technology degree in Computer Science and Engineering from the Indian Institute of Technology at Kanpur in May 1997. She was awarded the Motorola Student of the Year Gold Medal in May 1997. Thereafter, she received the Master of Sciences degree in Computer Sciences from the University of Texas at Austin in May 1999. She was awarded the MCD and J.C.Browne graduate fellowships in August 1997 and December 2001, respectively.

Permanent Address: c/o Narinder Singh Sehgal
2622 Phase 7, SAS Nagar
Sector 61, Chandigarh, India

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1]LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay and James A. Bednar.