Core-Stateless Guaranteed Rate Scheduling Algorithms*

Jasleen Kaur and **Harrick M. Vin**

Distributed Multimedia Computing Laboratory
Department of Computer Sciences
University of Texas at Austin

Abstract

Many per-flow scheduling algorithms have been proposed to provide rate and delay guarantees to flows. It is often argued that the need for maintaining per-flow state and performing per-packet classification seriously limits the scalability of routers that employ such per-flow scheduling algorithms. Consequently, design of algorithms that can provide per-flow rate and delay guarantees without requiring per-flow functionality in the network core routers has become an active area of research. In this paper, we propose a methodology to transform any *Guaranteed Rate* (GR) per-flow scheduling algorithm into a version that does not require per-flow state to be maintained in the core routers. We prove that a network of such *core-stateless* servers provides the same delay guarantee as a corresponding network of GR servers.

1 Introduction

The Internet has traditionally supported the *best-effort* service model in which the network offers no assurance about when, or even if, packets are delivered. This service model has proved to be adequate for elastic applications (e.g., ftp, telnet, and http) that tolerate packet delays and losses rather gracefully. With the commercialization of the Internet and the deployment of inelastic continuous media applications, however, the best-effort service model is increasingly becoming inadequate. For example, to meet the timeliness requirements of digital audio and video playback, most multimedia applications require greater predictability with respect to end-to-end delay and bandwidth than that offered by the current best-effort networks. To facilitate the co-existence of these emerging applications with the conventional elastic applications, there is an increasing need for designing networks that *differentiate* between the services provided to different customers and applications.

The *integrated services* network architecture achieves service differentiation by requiring (1) routers to employ per-flow[1] scheduling algorithms [2, 6, 12, 28] and (2) sources and destinations to exchange signaling messages that establish packet classification and forwarding state on each router along the path [21]. It is often argued that the need for maintaining per-flow state and performing per-packet classification in routers seriously limits the scalability of this architecture.

To address this scalability requirement, the *differentiated services* network architecture has been proposed [18]. In this architecture, traffic entering a network is classified and conditioned at the network boundary, and is assigned to a small set of behavior (or flow) aggregates (also referred to as Per Hop Behaviors—PHB). This architecture achieves scalability by implementing complex classification and conditioning functions only at network boundary routers (which process lower volumes of traffic and lesser numbers of flows), and providing service differentiation inside the network for flow aggregates rather than on a per-flow basis [18]. Simple resource-management mechanisms—such as RED [7]—have been proposed to provide service differentiation among behavior aggregates [4, 14, 15, 17]. While this architecture is inherently more scalable, it is less flexible and provides weaker guarantees to flows. In particular, it is non-trivial to provide per-flow rate or delay guarantees in this framework.

Recently, several frameworks and mechanisms have been proposed to overcome this tradeoff between scalability and flexibility [8, 24]. These mechanisms aim at providing service guarantees without requiring the core routers to either maintain per-flow state, or perform complex per-packet computation or both. In [8], simple buffer management schemes are used to provide rate guarantees to flows. This approach requires per-packet flow classification and maintaining per-flow state in routers, but eliminates the per-packet computation overhead inherent in per-flow scheduling algorithms. In [24], a *core-stateless* version of Jitter Virtual Clock (CJVC) 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. Since CJVC is non-work-conserving and 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-

*This research was supported in part by an AT&T Foundation Award, IBM Faculty Development Award, Intel, the National Science Foundation (CAREER award CCR-9624757, and Research Infrastructure Award CDA-9624082), Lucent Bell Laboratories, NASA, Mitsubishi Electric Research Laboratories (MERL), Tivoli, Novell, and Sun Microsystems Inc.

[1] A flow refers to a sequence of packets transmitted from a source to a destination.

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. Hence, stateless algorithms that provide delay guarantees and also are work-conserving are desirable.

In this paper, we extend the idea of core-stateless scheduling mechanisms to the entire class of *Guaranteed Rate* (GR) [10] per-flow scheduling algorithms. We describe a methodology using which any GR algorithm can be transformed into a core-stateless version, CSGR, that provides the same delay guarantees as the original GR algorithm. The network architecture we consider is similar to the DiffServ architecture, where per-flow functionality is implemented only at the edges of the network, and core routers do not maintain any per-flow state.

The rest of this paper is organized as follows. In Section 3, we show that the methodology in [24] for deriving the core-stateless version of Jitter Virtual Clock algorithm can not be directly applied to its work conserving counterpart, Virtual Clock. In Section 4, we observe the key property of GR algorithms which allows us to design the core-stateless version of Virtual Clock (CSVC). In Section 4.3, we prove that a network of CSVC servers has the same delay guarantee as a network of Virtual Clock servers. The methodology is generalized to all GR algorithms in Section 5. We discuss related work in Section 6 and summarize our results in Section 7.

2 Notation

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

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

Furthermore, we assume that the sum of rates reserved for flows at a server is less than the server capacity.

3 Problem Motivation

In [24], the authors develop the *Core-Stateless Jitter Virtual Clock* (CJVC) scheduling algorithm; CJVC eliminates the need for maintaining per-flow state in routers while providing the same delay guarantee as a network of Jitter Virtual Clock (JVC) servers. In this section, we ask the question: can the techniques used in deriving CJVC from JVC be applied directly to derive a core-stateless version of Virtual Clock—a work-conserving scheduling algorithm? 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:

$$e_{f,j}^1 = a_{f,j}^1 \tag{1}$$

$$e_{f,j}^k = \max\left(a_{f,j}^k + g_{f,j-1}^k, d_{f,j}^{k-1}\right) \tag{2}$$

$$d_{f,j}^k = e_{f,j}^k + \frac{l_f^k}{r_f^k}, \qquad j,k \geq 1 \tag{3}$$

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^k becomes 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 (2)). In deriving the Core-stateless Jitter Virtual Clock (CJVC) algorithm, the authors of [24] 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 $j \geq 2$:

$$\delta_f^k + a_{f,j}^k + g_{f,j-1}^k \geq d_{f,j}^{k-1} \tag{4}$$

In [24], 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 [29]—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:

$$VC_{f,j}^1 = a_{f,j}^1 + \frac{l_f^1}{r_f^1} \tag{5}$$

$$VC^k_{f,j} \quad = \quad \max(a^k_{f,j}, VC^{k-1}_{f,j}) + \frac{l^k_f}{r^k_f} \tag{6}$$

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 $j \geq 2$:

$$\delta^k_f \geq VC^{k-1}_{f,j} - a^k_{f,j} \tag{7}$$

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 [24]) 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) [10] 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.

4 Methodology

We begin by observing the following key property of *Guaranteed Rate* algorithms [10]: the upper bound on the deadline of a packet at a server can be expressed in terms of the deadline of the packet at the previous server; this state can be encoded in the packet itself (e.g., *Dynamic Packet State* [23]), thereby eliminating the need for maintaining per-flow state in core routers. We illustrate the use of this property in deriving a core-stateless version of the Virtual Clock algorithm, and prove the end-to-end delay guarantee of a network of such core-stateless VC servers.

4.1 Key Insight

The *Guaranteed Rate Clock* (GRC) of a per-flow scheduling algorithm is defined in [10] as:

$$GRC^1_{f,j} \quad = \quad a^1_{f,j} + \frac{l^1_f}{r^1_f} \tag{8}$$

$$GRC^k_{f,j} \quad = \quad \max(a^k_{f,j}, GRC^{k-1}_{f,j}) + \frac{l^k_f}{r^k_f} \tag{9}$$

where $GRC^k_{f,j}$ is the GRC of packet p^k_f at server j.

A scheduling algorithm at server j is defined to be a member of the class of *Guaranteed Rate* (GR) scheduling algorithms for flow f if it guarantees that p^k_f will be transmitted by $GRC^k_{f,j} + \beta_j$, where β_j is a constant that depends on the scheduling algorithm and the server. The class of GR servers is broad – it includes Virtual Clock, Packet-by-Packet Generalized Processor Sharing [20], Self Clocked Fair Queuing [9], Delay-EDD and Jitter-EDD [27]. Table 1 lists the values of β_j for some GR algorithms (derived in [11]).

The following lemma (stated and proved in [10]) provides the key property that we use in the design of core-stateless versions of Virtual Clock and other GR algorithms:

Lemma 1 *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.

The *guaranteed rate clock* (GRC) of a packet determines the *deadline* by which it gets transmitted. Lemma 1 provides an upper bound on the GRC of a packet at a server j in terms of the same packet's state at the previous server $j-1$. This state can be encoded in the packet itself at server $j-1$ and can be used for computing the upper bound on GRC at the server j. If the scheduler at server j uses this upper bound on GRC of a packet as its transmission deadline, it does not require any per-flow state to be maintained at server j.

We illustrate the above idea by applying it to define the *core-stateless* version of Virtual Clock, a work-conserving scheduling algorithm that belongs to GR. We generalize the methodology to all GR algorithms in Section 5.

4.2 Core Stateless Virtual Clock

The Virtual Clock algorithm guarantees that a packet would depart server j by $(VC^k_{f,j} + \frac{l^{max}_j}{C_j})$, where $VC^k_{f,j}$ is computed according to Equations (5) and (6), C_j is the capacity of server j and l^{max}_j the maximum size of packets it serves. From this, it is easy to see that Virtual Clock belongs to the class of GR algorithms with $GRC^k_{f,j} = VC^k_{f,j}$ and $\beta_j = \frac{l^{max}_j}{C_j}$.

From Lemma 1, we know that

$$VC^k_{f,j} \leq VC^k_{f,j-1} + \max_{i\in[1..k]} \frac{l^i_f}{r^i_f} + \pi_{j-1} + \frac{l^{max}_{j-1}}{C_{j-1}} \tag{10}$$

Define the *core virtual clock* of p^k_f at server j, $VCore^k_{f,j}$, as follows:

$$VCore^k_{f,1} \quad = \quad VC^k_{f,1} \tag{11}$$

GR algorithm	β_j
Virtual Clock	l_j^{max}/C_j
Packet-by-Packet GPS [20]	l_j^{max}/C_j
Self Clocked Fair Queuing [9]	$\sum_{m \neq f} l_j^{max}/C_j$
Delay EDD	$l_j^{max}/C_j + d_{f,j} - l_f/r_{f,j}$

Table 1: β_j values of some GR algorithms (For Delay EDD, $d_{f,j}$ is the desired delay bound for flow f at server j)

$$VCore_{f,j}^k = VCore_{f,j-1}^k + \beta_{j-1} + \pi_{j-1} + \max_{i \in [1..k]} \frac{l_f^i}{r_f^i}, \quad j \geq 2 \qquad (12)$$

where $\beta_{j-1} = \frac{l_{j-1}^{max}}{C_{j-1}}$. 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 (10)-(12), 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 (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 Delay Guarantee of a Network of CSVC Servers

Consider a network of J $CSVC$ servers. Let $L_{CSVC}^j(p_f^k)$ denote the time at which packet p_f^k departs server j. Then the end-to-end delay D_f^k of packet p_f^k is defined as:

$$D_f^k = L_{CSVC}^J(p_f^k) - a_{f,1}^k \qquad (13)$$

The delay guarantee of the network of servers is defined as the upper bound on D_f^k.

To prove the delay guarantee of a network of CSVC servers in Theorem 2, we first prove the delay bound of a single CSVC server (Theorem 1). 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 1, we first state and prove the following Lemmas.

Lemma 2 *The* core virtual clock *of packet p_f^k satisfies*

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

Proof: See Appendix B. ■

Lemma 3 *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*

$$L_{PCSVC}^j(p_f^k) \leq VCore_{f,j}^k \qquad j \geq 1 \qquad (14)$$

Proof: See Appendix C. ■

The following lemma is stated and proved in [11].

Lemma 4 *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 1 *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 $L_{CSVC}^j(p_f^k)$, is*

$$L_{CSVC}^j(p_f^k) \leq VCore_{f,j}^k + \frac{l_j^{max}}{C_j} \qquad (15)$$

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 1 follows immediately from Lemma 3 and Lemma 4. ■

Theorem 1 states that the deadline for the departure of packet

p_f^k at server j is $(VCore_{f,j}^k + \beta_j)$. To avoid errors in computation of $VCore_{f,j-1}^k$ and π_{j-1} due to inaccuracies in clock synchronization and imprecise knowledge of propagation delays, Equation (12) can be rewritten in terms of an alternate set of variables as:

$$VCore_{f,j}^k = a_{f,j}^k + g_{f,j-1}^k + \max_{i\in[1..k]} \frac{l_f^i}{r_f^i}, \quad j \geq 2 \quad (16)$$

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

Theorem 2 *The delay guarantee of a network of CSVC servers is the same as that of a network of Virtual Clock servers.*

Proof: Using Theorem 1, Equation (11) and repeated application of Equation (12), it follows that the time $L_{CSVC}^J(p_f^k)$ at which packet p_f^k departs the last server in a network of J CSVC servers is given by

$$\begin{aligned} L_{CSVC}^J(p_f^k) &\leq VCore_{f,J}^k + \beta_J \\ &\leq VC_{f,1}^k + (J-1)\max_{i\in[1..k]}\frac{l_f^i}{r_f^i} \\ &\quad + \sum_{i=1}^{J-1}(\beta_i + \pi_i) + \beta_J \end{aligned}$$

where $\beta_i = \frac{l_i^{max}}{C_i}$. From (13), the delay guarantee of the network of CSVC servers is thus given by:

$$\begin{aligned} D_f^k &\leq VC_{f,1}^k - a_{f,1}^k + (J-1)\max_{i\in[1..k]}\frac{l_f^i}{r_f^i} \\ &\quad + \sum_{i=1}^{J-1}(\beta_i + \pi_i) + \beta_J \end{aligned}$$

The delay guarantee of a network of J Virtual Clock servers has been shown to be the same as the above in [10]. ■

The quantity $(VC_{f,1}^k - a_{f,1}^k)$ in the above formulation depends on the source traffic characterization. For a Leaky Bucket source characterization [20], it can be shown [10] that

$$VC_{f,1}^k - a_{f,1}^k \leq \frac{\sigma_f}{r_f}$$

where σ_f is the depth of the Leaky Bucket and r_f is its average rate. If flow f conforms to a Leaky Bucket with parameters (σ_f, r_f), then its end-to-end delay is given by:

$$D_f^k \leq \frac{\sigma_f}{r_f} + (J-1)\max_{i\in[1..k]}\frac{l_f^i}{r_f^i} + \sum_{i=1}^{J-1}(\beta_i + \pi_i) + \beta_J$$

5 Core-Stateless GR Scheduling Algorithms

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

Define the *core guaranteed rate clock* ($GRCore$) for packet p_f^k at server j, $GRCore_{f,j}^k$, as follows:

$$\begin{aligned} GRCore_{f,1}^k &= GR_{f,1}^k \\ GRCore_{f,j+1}^k &= GRCore_{f,j}^k + \beta_j + \pi_j + \max_{i\in[1..k]}\frac{l_f^i}{r_f^i}, \\ & \qquad j \geq 1 \end{aligned}$$

For every GR algorithm, define a corresponding *Core-Stateless GR* algorithm that assigns $GRCore$ values to packets of all flows as defined above, and schedules packets in the increasing order of their $GRCore$ values.

Theorem 3 *The delay guarantee of a network of CSGR servers is the same as that of a network of corresponding GR servers.*

Proof: The proof of the above theorem can be constructed in exactly the same manner as the proof of Theorem 2 by replacing VC and $VCore$ by GRC and $GRCore$ respectively in the corresponding proof analysis of Lemmas 2 and 3, and Theorems 1 and 2. ■

From Theorem 3, 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 [5, 27].

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) [6] algorithm, then the resulting core-stateless algorithm will have the same delay property as WFQ, but would not guarantee fairness. In general, it is non-trivial to design core-stateless versions of fair scheduling algorithms. This is because the per-packet computation of a flow in a fair GR server depends on the other flows that are sharing the server. Hence this computation cannot be performed by the edge routers for a flow. Recently, schemes that achieve approximate fairness without maintaining per-flow state in network cores have been proposed [3, 19, 23].

6 Related Work

In the *differentiated services* architecture, per-flow classification and computation is performed only at the network edges – the network core maintains state for only a few *behavior aggregates* [4, 14, 15, 17]. While this architecture is inherently scalable, it is non-trivial to implement services that provide per-flow rate or delay guarantees.

In order to preserve the flexibility offered by per-flow management, recent research effort has focussed on reducing the complexity and increasing the efficiency of sorting and per-flow classification. Sorting complexity of $O(\text{loglog } n)$ has been reported in [25]. On the other hand, sorting is completely avoided in [8, 26]. In [8], simple buffer management schemes are used to provide rate guarantees – however, the buffer space is mostly statically partitioned, which results in poor resource utilization.

It is widely believed that per-flow classification is more complex than sorting. A number of schemes have been proposed and/or implemented that make flow classification more efficient [1, 13, 16, 22].

A different research theme assumes that the number of flows in the future Internet would be too large for the speedups in the existing per-flow mechanisms to suffice. Therefore, it focuses on completely eliminating per-flow state and per-packet classification in the network cores, and yet attempts to provide per-flow guarantees [5, 24]. The SCED+ algorithm proposed in [5] avoids the use of per-path state in order to provide guarantees to traffic aggregates that share the same path in an ATM network. However guarantees are not tailored to meet the specific requirements of individual flows. The CJVC algorithm [24] on the other hand provides per-flow delay guarantees without maintaining per-flow state in the network core. Additionally, by employing a CBR shaper at every router, the queues are not allowed to grow large, thus resulting in smaller sorting overhead. However, this scheme is non work-conserving, which limits the gains of *statistical multiplexing* it can benefit from.

7 Summary

In this paper we propose a methodology for transforming any *Guaranteed Rate* (GR) algorithm into its core-stateless counterpart, that eliminates the need for maintaining per-flow state or performing per-packet classification in core routers. We have shown that a network of servers running such *Core Stateless Guaranteed Rate* (CSGR) algorithms provides the same end-to-end delay guarantee as a network of corresponding GR servers.

References

[1] A. Begel, S. McCanne, and S. L. Graham. BPF+: Exploiting Global Data-flow Optimization in a Generalized Packet Filter Architecture. In *Proceedings ACM Sigcomm'99*, Sept 1999.

[2] 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.

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

[4] D. Clark and W. Fang. Explicit Allocation of Best Effort Packet Delivery Service. *IEEE/ACM Transactions on Networking*, 1(6):362–373, August 1998.

[5] R. L. Cruz. SCED+: Efficient Management of Quality of Service Guarantees. In *Proceedings of INFOCOM'98*, March 1998.

[6] 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.

[7] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. In *IEEE/ACM Transactions on Networking*, August 1993.

[8] R. Geurin, S. Kamat, V. Peris, and R. Rajan. Scalable QoS Provision Through Buffer Management. In *Proceedings ACM Sigcomm'98*, Sept 1998.

[9] S.J. Golestani. A Self-Clocked Fair Queueing Scheme for High Speed Applications. In *Proceedings of INFOCOM'94*, 1994.

[10] 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.

[11] P. Goyal and H. M. Vin. Generalized Guaranteed Rate Scheduling Algorithms: A Framework. Technical Report TR-95-30, Department of Computer Sciences, The University of Texas at Austin, 1995. Available via URL http://www.cs.utexas.edu/users/dmcl.

[12] 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.

[13] P. Gupta and N. McKeown. Packet Classification on Multiple Fields. In *Proceedings ACM Sigcomm'99*, Sept 1999.

[14] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. June 1999. Internet RFC 2597.

[15] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. August 1998. Internet Draft.

[16] T. V. Lakshman and D. Stiliadis. High-speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching. In *Proceedings ACM Sigcomm'98*, Sept 1998.

[17] K. Nichols, V. Jacobson, and L. Zhang. An Approach to Service Allocation in the Internet. November 1997. Internet Draft.

[18] K. Nichols, V. Jacobson, and L. Zhang. A Two-bit Differentiated Services Architecture for the Internet. November 1997. Internet Draft.

[19] 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.

[20] 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.

[21] 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.

[22] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast Scalable Algorithms for Level Four Switching. In *Proceedings ACM Sigcomm'98*, Sept 1998.

[23] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings ACM Sigcomm'98*, Sept 1998.

[24] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow Management. In *Proceedings ACM Sigcomm'99*, Sept 1999.

[25] S. Suri, G. Varghese, and G. Chandramenon. Leap Forward Virtual Clock: A New Fair Queueing Scheme with Guaranteed Delay and Throughput Fairness. In *Proceedings of INFOCOM'97*, April 1997.

[26] D. Wrege and J. Liebeherr. A Near-optimal Packet Scheduler for QoS Networks. In *Proceedings of INFOCOM'97*, April 1997.

[27] H. Zhang. Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10), October 1995.

[28] H. Zhang and S. Keshav. Comparison of Rate-Based Service Disciplines. In *Proceedings of ACM SIGCOMM*, pages 113–121, August 1991.

[29] L. Zhang. VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proceedings of ACM SIGCOMM'90*, pages 19–29, August 1990.

A Applying approach of [24] to Virtual Clock

We apply the approach adopted in [24] 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 J 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,

$$\delta_f^k \geq VC_{f,j}^{k-1} - a_{f,j}^k, \qquad j = 2, ..., J; k = 2, ..., K \quad (17)$$

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

$$VCore_{f,j}^k = a_{f,j}^k + \delta_f^k + \frac{l_f^k}{r_f^k}, j = 2, ..., J; k = 1, ..., K \quad (18)$$

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

Note that, from Equations (6), (17), and 18, we get:

$$VCore_{f,j}^k \geq VC_{f,j}^k, \qquad \forall j, k, f \quad (19)$$

Using 19, we can show that a server scheduling packets in increasing order of their *core virtual clock* values would transmit a packet by $(VCore_{f,j}^k + \beta_j)$, where $\beta_j = \frac{l_j^{max}}{C_j}$ (the proof is similar to the one for Theorem 1). Hence, the following upper bound on $a_{f,j}^k$ exists:

$$a_{f,j}^k \leq VCore_{f,j-1}^k + \beta_{j-1} + \pi_{j-1} \quad (20)$$

For computing δ_f^k, note that $\delta_f^1 = 0$.

Further, in order to ensure Inequality (17), we need to make sure that δ_f^k is no smaller than the largest value that its *RHS* can take. This is given by upper and lower bounds on $VC_{f,j}^{k-1}$ and $a_{f,j}^k$ respectively as follows:

$$\begin{aligned}
VC_{f,j}^{k-1} &= \max(a_{f,j}^{k-1}, VC_{f,j}^{k-2}) + \frac{l_f^{k-1}}{r_f^k} \\
&\leq a_{f,j}^{k-1} + \delta_f^{k-1} + \frac{l_f^{k-1}}{r_f^k} \qquad (from(19)) \\
&\leq VCore_{f,j-1}^{k-1} + \pi_{j-1} + \beta_{j-1} + \delta_f^{k-1} + \frac{l_f^{k-1}}{r_f^k} \\
&\qquad (from(20)) \\
&\leq a_{f,j-1}^{k-1} + 2(\delta_f^{k-1} + \frac{l_f^{k-1}}{r_f^k}) + (\pi_{j-1} + \beta_{j-1}) \\
&\quad . \\
&\quad . \\
&\leq a_{f,2}^{k-1} + (j-1)(\delta_f^{k-1} + \frac{l_f^{k-1}}{r_f^k}) + \sum_{i=2}^{j-1} (\pi_i + \beta_i) \\
&\leq VC_{f,1}^{k-1} + (j-1)(\delta_f^{k-1} + \frac{l_f^{k-1}}{r_f^k}) + \sum_{i=1}^{j-1} (\pi_i + \beta_i)
\end{aligned}$$

$$\begin{aligned}
a_{f,j}^k &\geq a_{f,j-1}^k + \pi_{j-1} + \frac{l_f^k}{C_j} \\
&\geq a_{f,j-2}^k + \sum_{i=j-2}^{j-1} (\pi_i + \frac{l_f^k}{C_i}) \\
&\quad . \\
&\quad . \\
&\geq a_{f,1}^k + \sum_{i=1}^{j-1} (\pi_i + \frac{l_f^k}{C_i})
\end{aligned}$$

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

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

Since $(\beta_i \geq l_f^k / C_i)$, the *RHS* is maximized for $j = J$, and therefore, the least sequence of δ_f^k that would satisfy Inequality (17) is given by:

$$\begin{aligned}
\delta_f^1 &= 0 \\
\delta_f^k &= max((0, (VC_{f,1}^{k-1} - a_{f,1}^k) + (J-1)(\delta_f^{k-1} + \frac{l_f^{k-1}}{r_f^k}) \\
&\quad + \sum_{i=1}^{J-1} (\beta_i - \frac{l_f^k}{C_i}))) \quad (21)
\end{aligned}$$

A.1 Delay Bound of a Network of Core-Stateless Virtual Clock Servers

Consider a flow f traveling through a network of J Core-Stateless Virtual Clock Servers. Let $L_{f,j}^k$ be the time at which packet p_f^k gets transmitted at the j^{th} node. Then, the bound on the end-to-end delay of the k^{th} packet is given by:

$$\begin{aligned}
L_{f,J}^k - a_{f,1}^k &\leq VCore_{f,J}^k + \beta_J - a_{f,1}^k \\
&\leq a_{f,J}^k - a_{f,1}^k + \delta_f^k + \frac{l_f^k}{r_f^k} + \beta_J \\
&\quad . \\
&\quad . \\
&\leq a_{f,2}^k - a_{f,1}^k + (J-1)(\delta_f^k + \frac{l_f^k}{r_f^k}) \\
&\quad + \sum_{i=2}^{J-1} (\pi_i + \beta_i) + \beta_J \\
L_{f,J}^k - a_{f,1}^k &\leq VC_{f,1}^k - a_{f,1}^k + (J-1)(\delta_f^k + \frac{l_f^k}{r_f^k}) \\
&\quad + \sum_{i=1}^{J-1} (\pi_i + \beta_i) + \beta_J
\end{aligned}$$

This delay bound 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 (21), we get:

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

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 bounds for even constant bit-rate flows, and hence the approach used in [24] does not work for Virtual Clock.

B Proof of Lemma 2

Observe that

$$\max_{i \in [1..k]} \frac{l_f^i}{r_f^i} \geq \max_{i \in [1..k-1]} \frac{l_f^i}{r_f^i} \quad (22)$$

Also observe that by repeatedly applying 12, we get:

$$\begin{aligned}
VCore_{f,j}^{k-1} &= VCore_{f,2}^{k-1} + (j-2) \max_{i \in [1..k-1]} \frac{l_f^i}{r_f^i} \\
&\quad + \sum_{i=2}^{j-1} (\beta_i + \pi_i) \\
&= VC_{f,1}^{k-1} + (j-1) \max_{i \in [1..k-1]} \frac{l_f^i}{r_f^i} \\
&\quad + \sum_{i=1}^{j-1} (\beta_i + \pi_i) \quad (23)
\end{aligned}$$

Similarly,

$$\begin{aligned}
VCore_{f,j}^k &= VC_{f,1}^k + (j-1) \max_{i \in [1..k]} \frac{l_f^i}{r_f^i} + \sum_{i=1}^{j-1} (\beta_i + \pi_i) \\
&\geq VC_{f,1}^{k-1} + \frac{l_f^{k-1}}{r_f^k} + (j-1) \max_{i \in [1..k]} \frac{l_f^i}{r_f^i} \\
&\quad + \sum_{i=1}^{j-1} (\beta_i + \pi_i) \quad (from (6)) \\
&\geq VC_{f,1}^{k-1} + \frac{l_f^{k-1}}{r_f^k} + (j-1) \max_{i \in [1..k-1]} \frac{l_f^i}{r_f^i}
\end{aligned}$$

$$+\sum_{i=1}^{j-1}(\beta_i+\pi_i) \quad (from(22))$$

$$\geq \quad VCore_{f,j}^{k-1}+\frac{l_f^{k-1}}{r_f^k} \qquad (from(23))$$

C Proof of Lemma 3

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

$$R_{f,j}(t)=\begin{cases} r_{f,j}^k & \text{if } \exists k \ni (a_{f,j}^k \leq t) \wedge \\ & \quad (VCore_{f,j}^{k-1} < t \leq VCore_{f,j}^k) \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

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 3 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 [11].

Induction Hypothesis : Assume 14 holds for $1 \leq j \leq m$.

Induction : We will show that 14 holds for $1 \leq j \leq m+1$. From (16) and the Induction Hypothesis (which implies that $g_{f,m}^k \geq 0$) we get

$$VCore_{f,m+1}^k \quad \geq \quad a_{f,m+1}^k+\frac{l_f^k}{r_f^k} \tag{25}$$

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_f^k, let $T_{f,j}^k$ denote the following quantity

$$T_{f,j}^k = VCore_{f,j}^k - \max{(a_{f,j}^k, VCore_{f,j}^{k-1})}$$

It is easily observed from (24), (25) and Lemma 2 that

$$\begin{aligned}\int_{t_1}^{t_2} R_{f,m+1}(t)dt &= \sum_{i\in K_f(t_1,t_2)} (r_{f,m+1}^i * T_{f,m+1}^i) \\ &\geq \sum_{i\in K_f(t_1,t_2)} (r_{f,m+1}^i * \frac{l_f^i}{r_f^i}) \\ &\geq \sum_{i\in K_f(t_1,t_2)} l_f^i\end{aligned}$$

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

$$AP_f(t_1,t_2) \leq \int_{t_1}^{t_2} R_{f,j}(t)dt \tag{26}$$

We now prove the theorem 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

$$\begin{aligned}L_P^{m+1}CSVC(p_f^k) &= a_{f,m+1}^k+\frac{l_f^k}{C_{m+1}} \\ &\leq VCore_{f,m+1}^k\end{aligned}$$

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:

$$\begin{aligned}\sum_{f\in S} AP_f(t_1,t_2) &> C_{m+1}(t_2-t_1) \\ \sum_{f\in S}\int_{t_1}^{t_2} R_{f,m+1}(t)dt &> C_{m+1}(t_2-t_1) \\ \int_{t_1}^{t_2}\sum_{f\in S} R_{f,m+1}(t)dt &> C_{m+1}(t_2-t_1)\end{aligned} \tag{27}$$

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 (27) and hence the induction step is proved. From induction, the theorem follows.