# BCSQ: Bin-based Core Stateless Queueing for Scalable Support of Guaranteed Services

Zhenhai Duan and Karthik Parsha
Computer Science Department
Florida State University
Tallahassee, FL 32306
Email: {duan, parsha}@cs.fsu.edu

***Abstract*— Core stateless packet scheduling systems have received considerable attention in recent years because of their scalability in supporting per-flow Quality of Services guarantees. In such a system core routers do not need to maintain per-flow state and do not need to perform per-flow operations such as per-flow classification, per-flow queueing, and per-flow scheduling. On the other hand, existing core stateless packet schedulers require core routers to sort incoming packets based on their virtual finish times. This sorting operation results in the worst-case runtime complexity of $O(\log_2 N)$, where $N$ is the number of packets in a scheduler. In this paper we propose a bin-based core stateless queueing (BCSQ) algorithm, which achieves constant runtime complexity that is independent of the number of packets in the scheduler. We present the detailed design of BCSQ and derive the worst-case end-to-end delay bounds for packets in a network of BCSQ. In addition, we investigate the effects of the configurable parameters of BCSQ on the performance of BCSQ networks. Simulation studies are also performed to illustrate the efficacy and performance of BCSQ.**

## I. Introduction

Core stateless packet scheduling systems have a number of important advantages in providing scalable and flexible support of per-flow Quality of Service (QoS) guarantees on the Internet. In addition to eliminating the need for maintaining per-flow scheduling state at core routers, such systems also relieve core routers of performing per-flow operations including per-flow packet classification, per-flow queueing, and per-flow scheduling. Moreover, they also help decouple QoS control plane functions such as admission control from the packet forwarding data plane on core routers. The QoS control plane functions can now be implemented in separate network entities such as bandwidth brokers [2], [9], [17], [19]. As a result, core stateless systems have received considerable attention in recent years (e.g., [8], [10], [11], [12], [13], [15], [20]), and many core stateless packet schedulers/architectures have been proposed, for example, Core Jitter Virtual Clock (CJVC) [15], Virtual Time Reference Systems (VTRS) [20], Core Stateless Guaranteed Rate Algorithms (CSGR) [10], and Coordinated Network Scheduling (CNS) [13].

Core stateless systems rely on the notion of *dynamic packet state* to achieve the above advantages [16]. In a network of core stateless packet schedulers, core routers are distinguished from edge routers; whereas edge routers maintain per-flow state and perform per-flow operations, core routers do not. Core routers schedule packets based on dynamic packet state—virtual finish time—carried in the packet headers. The packet with the smallest virtual finish time departs from the scheduler first. Conceptually, the virtual finish time of a packet in a core stateless packet scheduler emulates the real departure time of the packet in a reference stateful packet scheduler. However, the computation of virtual finish times is core stateless in that core routers do not need to maintain per-flow state or perform per-flow operations in order to compute the virtual finish time of an incoming packet. On the other hand, although core stateless systems eliminate the need for per-flow state maintenance and per-flow operations at core routers, core routers need to sort packets according to their virtual finish times. The sorting operation has the worst-case runtime complexity of $O(\log_2 N)$, where $N$ is the number of packets in the scheduler [18]. This may not be acceptable in a high-speed core router.

We propose a bin-based core stateless queueing (BCSQ) algorithm to overcome this problem. Conceptually, a BCSQ scheduler partitions the virtual time space into slots or bins of equal time intervals (see Fig. 2). A packet is placed into a bin if its virtual finish time falls in the time interval of the bin. Bins are ordered according to the time intervals they represent, and they are served in that order. Importantly, packets in a bin are served in a FIFO manner. Like the bin-based event scheduling algorithm *Calendar Queue* [5] and stateful bin sort fair queueing *BSFQ* [6], the proposed BCSQ scheduler achieves constant runtime complexity that is independent of the number of packets (or flows) in the scheduler.

In this paper we present the detailed design of BCSQ and derive the worst-case end-to-end delay bounds for packets in a network of BCSQ. We also study the effects of the number of bins and the length of time intervals on the performance of BCSQ networks. In particular, we investigate the minimum number of bins that a scheduler needs to maintain in order to prevent packet overflows due to the limited time window the bins can collectively represent. Moreover, simulation studies are performed to illustrate the efficacy and performance of BCSQ. Through simulations we show that, by controlling the length of time intervals the bins represent, BCSQ can achieve various performance and complexity trade-offs. For example, when time intervals are sufficiently long, all incoming packets will fall in a single bin and BCSQ degenerates into a FIFO scheduler. On the other hand, as we gradually decrease the

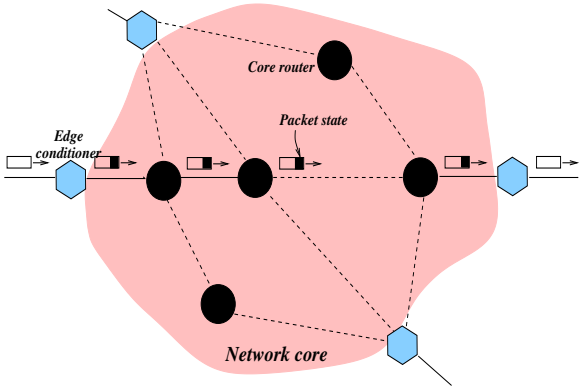


Fig. 1. Illustration of the virtual time reference system.

length of time intervals, BCSQ is able to provide improved per-flow QoS guarantees, albeit with greater scheduling complexity.

The rest of the paper is structured as follows. Section II presents a brief overview of the virtual time reference system (VTRS) framework [20]. BCSQ is developed within this framework. We describe the BCSQ scheduler and analyze its properties in Section III. We report simulation results in Section IV and conclude the paper in Section V.

## II. Background: Virtual Time Reference System

In this section we provide an overview of the virtual time reference system (VTRS), which was proposed in [20] as a unifying core stateless scheduling framework to provide scalable support for guaranteed services. In the next section we will present the new bin-based core stateless queueing algorithm (BCSQ). BCSQ is developed within the VTRS framework. Table I summarizes the notations used in the paper.

Like in other core stateless packet scheduling systems, core routers are distinguished from edge routers in the virtual time reference system. While edge routers maintain per-flow state and perform per-flow operations, core routers do not. VTRS consists of three logical components (Fig. 1): *edge traffic conditioning* at the network edge, *packet state* carried by packets, and *per-hop virtual time reference/update mechanism* at core routers. These three components are briefly described below.

### A. Edge Traffic Conditioning

Edge traffic conditioning plays a key role in VTRS, as it ensures that the packets of a flow[1] will never be injected into the network core at a rate exceeding its reserved rate. Formally, for a flow $j$ with a reserved rate $r^j$, the inter-arrival time of two consecutive packets of the flow at the first hop core router is such that $\hat{a}_1^{j,k+1} - \hat{a}_1^{j,k} \geq \frac{L^{j,k+1}}{r^j}$, where $\hat{a}_1^{j,k}$ denotes the arrival time[2] of the $k$th packet $p^{j,k}$ of flow $j$ at the network core and $L^{j,k}$ the size of packet $p^{j,k}$.

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

| | |
|---|---|
| $p^{j,k}$ | the $k$th packet of flow $j$ |
| $L^{j,k}$ | packet length of $p^{j,k}$ |
| $L^{j,max}$ | maximum packet length of flow $j$ |
| $L^{*,max}$ | maximum packet length of all flows at a node |
| $r^j$ | reserved rate of flow $j$ |
| $h$ | number of hops along the path of flow $j$ |
| $\Delta^{j,k}$ | cumulative queueing delay packet $p^{j,k}$ experienced along the path of flow $j$ |
| $\delta^{j,k}$ | virtual time adjustment term for packet $p^{j,k}$ |
| $\tilde{\omega}_i^{j,k}$ | virtual time stamp of packet $p^{j,k}$ at node $i$ |
| $\tilde{\nu}_i^{j,k}$ | virtual finish time of packet $p^{j,k}$ at node $i$ |
| $\tilde{d}_i^{j,k}$ | virtual delay of packet $p^{j,k}$ at node $i$ |
| $\hat{a}_i^{j,k}$ | actual time packet $p^{j,k}$ arrives at node $i$ |
| $\hat{f}_i^{j,k}$ | actual time packet $p^{j,k}$ departs from node $i$ |
| $\Psi_i$ | error term of the scheduler at node $i$ |
| $C_i$ | link capacity of node $i$ |
| $\pi_{i,i+1}$ | propagation delay from $i^{th}$ node to $(i+1)^{th}$ node |
| $M$ | Number of bins at a node |
| $N$ | Number of packets at a node |
| $\iota_i$ | length of bins at node $i$ |

TABLE I
Notation used in the paper.

### B. Packet State

After going through the edge conditioner at the network edge, packets entering the network core carry in their packet headers certain packet state information that is initialized and inserted at the network edge. The packet state carried by the $k$th packet $p^{j,k}$ of a flow $j$ contains three types of information: 1) QoS reservation (e.g., the reservation rate $r^j$) of the flow; 2) the virtual time stamp $\tilde{\omega}_i^{j,k}$ of the packet that is associated with the router $i$ currently being traversed; and 3) the virtual time adjustment term $\delta^{j,k}$ of the packet (see below). For the $k$th packet of flow $j$, its virtual time stamp $\tilde{\omega}_1^{j,k}$ is initialized to $\hat{a}_1^{j,k}$, the actual time it leaves the edge conditioner and enters the first core router along the flow's path. The virtual time adjustment term $\delta^{j,k}$ for packet $p^{j,k}$ is set to $\Delta^{j,k}/h$, where $h$ is the number of core routers along the flow's path, and $\Delta^{j,k}$ is computed at the network edge using the following recursive formula:

$$\begin{aligned} \Delta^{j,1} &= 0 \text{ and} \\ \Delta^{j,k} &= \max\left\{0, \Delta^{j,k-1} + h\frac{L^{j,k-1} - L^{j,k}}{r^j} + \hat{a}_1^{j,k-1} - \hat{a}_1^{j,k} + \frac{L^{j,k}}{r^j}\right\}, \\ &\quad \text{for } k = 2, 3, \ldots. \end{aligned}$$

The physical meaning of $\Delta^{j,k}$ is that it represents the cumulative queueing delay experienced by packet $p^{j,k}$ in an *ideal dedicated per-flow system*, where packets of flow $j$ are serviced by $h$ tandem servers with capacity $r^j$ [20].

[2]A packet is considered to have arrived at a server only when its last bit has been received, and it to have departed the server only when its last bit has been serviced. In addition, we assume that the edge conditioner and the first-hop router (i.e. the first core router) are cohabited, and thus the propagation delay from the edge conditioner to the first core router is negligible.

### C. Per-Hop Virtual Time Reference/Update Mechanisms at Core Routers

In the *conceptual* framework of the virtual time reference system, each core router is equipped with a per-hop virtual time reference/update mechanism to maintain the continual progression of the *virtual time* embodied by the packet virtual time stamps. This virtual time stamp $\tilde{\omega}_i^{j,k}$ represents the arrival time of the $k$th packet $p^{j,k}$ of flow $j$ at the $i$th core router *in the virtual time*, and thus it is also referred to as the *virtual arrival time* of the packet at the core router. The virtual time stamps $\tilde{\omega}_i^{j,k}$'s associated with packets of flow $j$ satisfy the following two important properties: 1) *virtual spacing property:* $\tilde{\omega}_i^{j,k+1} - \tilde{\omega}_i^{j,k} \geq \frac{L^{j,k+1}}{r^j}$, and 2) the *reality check property:* $\hat{a}_i^{j,k} \leq \tilde{\omega}_i^{j,k}$, where $\hat{a}_i^{j,k}$ denotes the actual arrival time of packet $p^{j,k}$ at router $i$. These two properties are important in ensuring that the end-to-end delay experienced by packets of a flow across the network core is bounded.

In order to ensure that these two properties are satisfied, the virtual time stamps must be appropriately referenced or updated as packets enter or leave a core router. The referencing/updating rule depends on the scheduling algorithm (or *scheduler*) employed by a core router and its characteristics. We associate two parameters with each arriving packet at a core scheduler: the *virtual delay* parameter and *virtual finish time*. They are computed as follows. Consider a packet $p^{j,k}$ arriving at the $i$th router along the path, then the virtual delay parameter for the packet is $\tilde{d}_i^{j,k} = L^{j,k}/r^j + \delta^{j,k}$, and its virtual finish time is defined as $\tilde{\nu}_i^{j,k} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k}$.

The *per-hop behavior* of a core router (or rather, its scheduler) is characterized by an *error term*, which is defined with respect to the virtual finish time and *actual* finish time of packets at the router. Let $\hat{f}_i^{j,k}$ denote the actual time packet $p^{j,k}$ departs the scheduler $i$. We say that scheduler $i$ can guarantee flow $j$ its reserved rate $r^j$ with an error term $\Psi_i$, if for any $k$, $\hat{f}_i^{j,k} \leq \tilde{\nu}_i^{j,k} + \Psi_i$. In other words, each packet of flow $j$ is guaranteed to depart $\mathcal{S}_i$ by the time $\tilde{\nu}_i^{j,k} + \Psi_i = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k} + \Psi_i$.

Given the error term $\Psi_i$ of the scheduler $i$, the virtual time stamp of packet $p^{j,k}$ after it has traversed $i$ is updated using the following reference/update rule:

$$\tilde{\omega}_{i+1}^{j,k} = \tilde{\nu}_i^{j,k} + \Psi_i + \pi_{i,i+1} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k} + \Psi_i + \pi_{i,i+1} \quad (1)$$

where $\pi_{i,i+1}$ denotes the propagation delay from the $i$th router to the next-hop router along the flow's path. In [20] it is shown that using the reference/update rule in (1) the virtual spacing and reality check properties of virtual time stamps are satisfied at every router.

### D. End-to-End Delay Bound of Packets in a VTRS Network

An important consequence of the virtual time reference system outlined above is that the bound on the end-to-end delay in a VTRS network experienced by packets of a flow across the network core can be expressed in terms of the reservation rate of a flow and the error terms of the routers along the flow's path. The following theorem concerning the end-to-end delay bound of packets in a VTRS network was proved in [20].

*Theorem 1:* Suppose there are total $h$ hops along the path of flow $j$, then for each packet $p^{j,k}$ of flow $j$, we have

$$\hat{f}_h^{j,k} - \hat{a}_1^{j,k} \leq h\frac{L^{j,max}}{r^j} + \sum_{i=1}^{h}\Psi_i + \sum_{i=1}^{h-1}\pi_{i,i+1} \quad (2)$$

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

Observe that the end-to-end delay formula (2) is quite similar to that specified in the IETF Guaranteed Service [4] using the WFQ [7] as the reference system. In this sense, the virtual time reference system provides a conceptual *core stateless* framework based on which guaranteed services can be implemented in a scalable manner using the DiffServ paradigm [1], [3].

A core stateless virtual clock scheduler (CSVC) was designed and studied in [20]. CSVC serves packets in the order of their virtual finish times. For any packet $p^{j,k}$ traversing scheduler $i$, let $\tilde{\omega}^{j,k}$ be the virtual time carried by $p^{j,k}$ as it enters scheduler $i$, and $\tilde{d}^{j,k} = \frac{L^{j,k}}{r^j} + \delta^{j,k}$ be its virtual delay. Then the virtual finish time $\tilde{\nu}^{j,k}$ of $p^{j,k}$ is given by $\tilde{\omega}^{j,k} + \tilde{d}^{j,k}$. Throughout the paper, we denote by $L^{*,max}$ the maximum size of packets of all flows traversing a core scheduler. The following Virtual Rate Control lemma was proved in [20].

*Lemma 2 (Virtual Rate Control Lemma):* Consider an arbitrary time interval $[\tau, t]$. We say that packet $p^{j,k}$ of flow $j$ is *virtually eligible* for service during $[\tau, t]$ if $\tilde{\omega}^{j,k} \geq \tau$ and $\tilde{\nu}^{j,k} \leq t$. Let $\tilde{S}^j$ denote the set of the packets of flow $j$ which are virtually eligible for service in $[\tau, t]$. Define $\tilde{W}^j(\tau, t) = \sum_{k\in\tilde{S}^j} L^k$. We refer to $\tilde{W}^j(\tau, t)$ as the *virtual eligible work* of flow $j$ over $[\tau, t)$. Then

$$\tilde{W}^j(\tau, t) = \sum_{k\in\tilde{S}^j} L^{j,k} \leq r^j(t - \tau). \quad (3)$$

## III. BCSQ: Bin-Based Core Stateless Queueing

In this section we present the new bin-based core stateless queueing algorithm BCSQ, which can be deployed at the core routers in the virtual time reference system framework. To simplify the presentation, we first assume that a BCSQ scheduler has an infinite number of bins, and derive the worst-case end-to-end delay bounds for packets in a BCSQ network. Then we study the practical scenario where BCSQ only has a finite number of bins and investigate the minimum number of bins that a BCSQ needs to maintain in order to avoid packet overflow due to the limited time window that the bins can collectively represent. We discuss the runtime scheduling complexity of BCSQ at the end of the section.

### A. BCSQ with Infinite Number of Bins

Conceptually, a BCSQ scheduler $i$ divides the virtual time space into slots or bins of equal time intervals of length $\iota_i$ (Fig. 2). Let $\tau_m = m\iota_i$, for $m = 0, 1, 2, \ldots$. Each bin $m$ represents the time interval $[\tau_m, \tau_{m+1})$, and has an associated FIFO queue with an unlimited buffer space; that is, there is no packet loss due to buffer overflow. In the following, we use

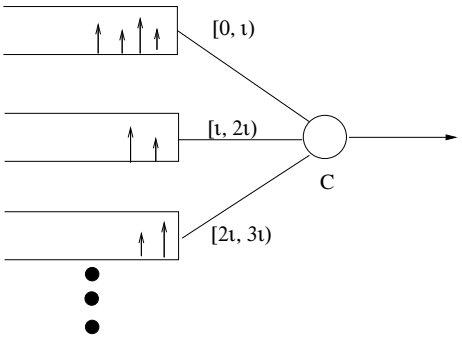


Fig. 2. Illustration of BCSQ.

the terms *bin* and its associated *queue* interchangeably. The scheduler has an infinite number of bins, i.e., $m \to \infty$. When the $k$th packet $p^{j,k}$ of flow $j$ arrives at a BCSQ scheduler, the scheduler will assign a virtual finish time $\tilde{\nu}^{j,k} = \tilde{\omega}^{j,k} + \tilde{d}^{j,k}$ to the packet, where $\tilde{\omega}^{j,k}$ and $\tilde{d}^{j,k}$ are the virtual arrival time and virtual delay of the packet, respectively. Like the core stateless virtual clock scheduler (CSVC), BCSQ assigns the packet the virtual delay using the following formula: $\tilde{d}^{j,k} = \frac{L^{j,k}}{r^j} + \delta^{j,k}$. The packet is placed in the queue associated with bin $m$ if $\tau_m \leq \tilde{\nu}^{j,k} < \tau_{m+1}$. The bins are serviced in the order of $\tau_m$. If bin $m$ is empty, the next non-empty bin is serviced. However, suppose a new packet arrives at a previously empty bin $m$ while a packet from a queue $m'$ where $\tau_{m'} > \tau_m$ is being serviced, the new packet has to wait. After the current packet being served by the scheduler departs, bin $m$ will be serviced next. When a packet $p^{j,k}$ departs from the scheduler $i$, the virtual time stamp carried in the packet header is updated using Eq. (1). We derive the error term of a BCSQ scheduler shortly.

To have a finite end-to-end delay bound for packets in a BCSQ network, the following admission control condition needs to hold at any BCSQ scheduler $i$ in the network. Assuming $F$ flows traversing scheduler $i$ with link capacity of $C_i$, then $\sum_{j=1}^{F} r^j \leq C_i$. The next lemma states the error term associated with a BCSQ scheduler $i$.

*Lemma 3:* Consider $F$ flows traversing a BCSQ scheduler $i$ such that the admission control condition $\sum_{j=1}^{F} r^j \leq C_i$ is satisfied. Suppose that $\hat{a}^{j,k} \leq \tilde{\omega}^{j,k}$ for any packet $p^{j,k}$ of flow $j$, $j = 1, 2, \ldots, F$. Then

$$\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \frac{L^{*,max}}{C_{\mathcal{S}}} + \iota_i. \tag{4}$$

In other words, the error term of a BCSQ scheduler $\Psi_i = \frac{L^{*,max}}{C_i} + \iota_i$.

*Proof:* Consider an arbitrary packet $p^{j,k}$ from a flow $j$, and the system busy period containing packet $p^{j,k}$. Suppose that $\tau_m \leq \tilde{\nu}^{j,k} < \tau_{m+1}$. Without loss of generality, we assume that the system busy period starts at time 0. Let $\tau$ be the last time before the arrival of packet $p^{j,k}$ such that the scheduler starts servicing a packet from a queue $m'$ such as $\tau_{m'} > \tau_m$. If such a packet does not exist, set $\tau = 0$, the beginning of the busy period. Hence $0 \leq \tau \leq \hat{a}^{j,k}$. For each flow $n$, $n = 1, 2 \ldots, F$, let $S^n$ denote the set of packets of flow $n$ that are serviced after $\tau$ and no later than packet $p^{j,k}$ (i.e., they are serviced during the time interval $(\tau, \hat{f}^{j,k}]$). From the definition of $\tau$, we see that for any $p^{n,l} \in S^n$, $\tilde{\omega}^{n,l} \geq \hat{a}^{n,l} \geq \tau$ and $\tilde{\nu}^{n,l} < \tau_{m+1} = \tau_m + \iota_i$. Applying the Virtual Rate Control Lemma (*Lemma* 2) to flow $n$ over the time interval $[\tau, \tau_m + \iota_i]$, we have

$$\sum_{l \in S^n} L^{n,l} \leq \tilde{W}^n(\tau, \tau_m + \iota_i) \leq r^n(\tau_m + \iota_i - \tau).$$

Summing over all $n$ and using the admission control condition, we have

$$\sum_{n=1}^{F} \sum_{l \in S^n} L^{n,l} \leq (\sum_{n=1}^{F} r^n)(\tau_m + \iota_i - \tau) \leq C_i(\tau_m + \iota_i - \tau).$$

Note that packet $p^{j,k} \in S^j$. Hence packet $p^{j,k}$ must depart the scheduler after all the packets in $\cup_{n=1}^{F} S^n$ have been serviced by the server. Furthermore, the packet which is being serviced at time $\tau$ (if it exists) has a size of at most $L^{*,max}$. Therefore,

$$\begin{aligned} \hat{f}^{j,k} &\leq \tau + \frac{L^{*,max}}{C_i} + \frac{\sum_{n=1}^{F} \sum_{l \in S^n} L^{n,l}}{C_i} \\ &\leq \tau_m + \iota_i + \frac{L^{*,max}}{C_i} \leq \tilde{\nu}^{j,k} + \iota_i + \frac{L^{*,max}}{C_i}. \end{aligned} \tag{5}$$

That is, the *error term* of a BCSQ scheduler $i$ is $\Psi_i = \frac{L^{*,max}}{C_i} + \iota_i$. ∎

The following theorem states the worst-case end-to-end delay bounds for packets in a network of BCSQ schedulers.

*Theorem 4:* Consider a flow $j$ that traverses a path of $h$ hops in a BCSQ network. BCSQ scheduler $i_i$ has a bin time interval of length $\iota_i$ and a link capacity of $C_i$, for $i = 1, 2, \ldots, h$. Then for any packet $p^{j,k}$ of flow $j$, its end-to-end delay in the network is bounded by:

$$\hat{f}_h^{j,k} - \hat{a}_1^{j,k} \leq h\frac{L^{j,max}}{r^j} + \sum_{i=1}^{h}(\frac{L_i^{*,max}}{C_i} + \iota_i) + \sum_{i=1}^{h-1} \pi_{i,i+1}, \tag{6}$$

*Proof:* The proof is trivial by noticing Lemma 3 and Eq.(2). ∎

### B. BCSQ with Finite Number of Bins

It is intuitively clear that a BCSQ scheduler does not need to maintain an infinite number of bins. When a bin with a lower time interval becomes empty, it can be reused for a higher time interval. Consider a scheduler $i$ with a number of $M$ bins. Figs. 3 and 4 present the pseudo-code for enqueueing and dequeueing a packet, respectively. As shown in Fig. 3, when a packet $p^{j,k}$ arrives at the scheduler, the scheduler first computes the virtual finish time $\tilde{\nu}^{j,k}$ of the packet, and determines the bin number $m$ that the packet should enter: $m = \lfloor (\tilde{\nu}^{j,k} - \tau_0)/\iota_i \rfloor$ (lines 1-2). If such a bin exists (line 3-7), the scheduler appends the new packet to the tail of the corresponding queue. If the index of the queue $m$ is smaller than the $currentqueue$ queue that the scheduler is currently serving, the index $currentqueue$ is updated to the new queue $m$.

```
0.  Packet p^{j,k} arrives at scheduler i;
1.  ν̃^{j,k} = ω̃^{j,k} + d^{j,k};
2.  m = ⌊(ν̃^{j,k} − τ_0)/ι_i⌋;
3.  if (0 ≤ m < M)
4.     Add p^{j,k} to the tail of queue m;
5.     if m < currentqueue
6.        currentqueue = m
7.     end if
8.  else if (m ≥ M)
9.     Rotate bins 0 to m − M to (new) highest time intervals;
10.    Add p^{j,k} to the tail of (new) queue M − 1;
11. else /* m < 0 */
12.    Rotate bins M − 1 to M − |m| to (new) lowest time intervals;
13.    Add p^{j,k} to the tail of (new) queue 0;
14.    currentqueue = 0
15. end if
```

Fig. 3. Pseudo-code for enqueueing a packet.

```
0.  Scheduler i finished serving the current packet;
1.  if (queue currentqueue ≠ ⊥)
2.     p = dequeue HOL of queue currentqueue;
3.  else
4.     currentqueue = next_nonempty_queue();
5.     if (currentqueue ≥ 0)
6.        p = dequeue HOL of queue currentqueue;
7.     end if
8.  end if
9.  if (p ≠ ⊥)
10.    ω̃^{j,k} = ν̃^{j,k} + Ψ_i + π_{i,i+1}
11.    Serving packet p;
12. end if
```

Fig. 4. Pseudo-code for dequeueing a packet.

If the packet has a large virtual finish time that is not covered by the time window collectively represented by bins 0 to $M-1$, i.e., $\tilde{\nu}^{j,k} \geq \tau_M$ (line 8-10), the scheduler will rotate $m - M + 1$ bins from the lowest time intervals to the (new) highest time intervals. For the time being, we assume that $M$ is sufficiently large that the lowest $m - M + 1$ queues are empty when the bin rotation is performed. We will investigate the minimum number of bins required to ensure such a condition be always held shortly. The scheduler then updates the bin indexes and appends the new packet to the tail of the queue $M - 1$. Similarly, if the virtual finish time of the packet is smaller than $\tau_0$ (line 11-14), the scheduler will rotate $|m|$ bins from the highest time intervals to the (new) lowest time intervals. Again we assume that $M$ is large enough so that the queues with the $|m|$ highest time intervals are empty when the rotation is performed. Then the scheduler adds the new packet to queue 0 and updates $currentqueue$ to 0 accordingly.

Fig. 4 illustrates the procedure for the scheduler to depart a packet. After the scheduler finishes serving the current packet, it checks if the current queue $currentqueue$ is empty (line 1-3). If the queue is not empty, the head-of-line packet is dequeued from the current queue. Otherwise, the scheduler searches the next nonempty queue (line 5-8), and dequeues the head-of-line packet of the queue if such a queue is found. Otherwise, there is no packet to be served and the scheduler becomes idle. When the next packet to be transmitted is found (line 9-12), scheduler $i$ updates the virtual time stamp of the packet and transmits the packet accordingly.

*1) Number of Bins:* The number of bins $M$ and the time interval $\iota_i$ of scheduler $i$ must be properly configured so that all incoming packets can be placed into one of the bins (assuming the amount of traffic released into the network from all flows has been properly conditioned at the network edge). We use the following concepts to derive a condition that $M$ and $\iota_i$ need to satisfy, in order to avoid packet overflow due to the limited time window that is collectively represented by the bins in the scheduler. We refer to $\Theta_i = M\iota_i$ as the *virtual time window size* of $i$. Denote by $P_i$ the set of packets in the scheduler when an arbitrary packet $p^{j,k}$ from flow $j$ arrives. In order for the packet to be placed in one of the queues, the scheduler parameters $M$ and $\iota_i$ must be configured in a way that the following condition holds: $|\tilde{\nu}^{j,k} - \tilde{\nu}^{n,l}| < \Theta_i$, for any packet $p^{n,l} \in P_i$. We denote by $\tilde{\nu}^{j,k} \in \Theta_i$ the fact that packet $p^{j,k}$ can be placed into one of the queues of scheduler $i$.

*Theorem 5:* Let $D$ be the largest worst-case end-to-end delay bound of any flow in a BCSQ network. Let $\Theta_i$ be the virtual time window size of a BCSQ scheduler $i$ in the network and assume that the empty bin with the lowest or highest time interval can be reused. If $\Theta_i \geq 2D$, for any incoming packet $p^{j,k}$ of an arbitrary flow $j$, we have $\tilde{\nu}_i^{j,k} \in \Theta_i$. That is, all incoming packets can be placed into a proper bin at the scheduler.

*Proof:* Consider an arbitrary packet $p^{j,k}$ from flow $j$ arriving at the BCSQ scheduler $i$. Without loss of generality, we assume that $\tilde{\nu}_i^{j,k}$ is larger than the virtual finish times of all other packets in the scheduler when packet $p^{j,k}$ arrives. Let $p^{n,l}$ be the packet with the smallest virtual finish time currently in the scheduler. To prove the theorem, we simply need to establish that $\tilde{\nu}_i^{j,k} - \tilde{\nu}_i^{n,l} \leq 2D$. We prove this by contradiction.

Assume that $\tilde{\nu}_i^{j,k} - \tilde{\nu}_i^{n,l} > 2D$. Notice $\tilde{\nu}_i^{n,l} \geq \hat{a}_1^{n,l}$, we know that $p^{n,l}$ must have entered the network before $\tilde{\nu}_i^{j,k} - 2D$. That is

$$\hat{a}_1^{n,l} < \tilde{\nu}_i^{j,k} - 2D, \tag{7}$$

otherwise we have $\tilde{\nu}_i^{j,k} - \tilde{\nu}_i^{n,l} \leq \tilde{\nu}_i^{j,k} - \hat{a}_1^{n,l} \leq 2D$, we reach a contradiction. Note that the *real* time when $p^{j,k}$ arrives at scheduler $i$ is $\hat{a}_S^{j,k}$, which is no less than $\hat{a}_1^{j,k}$, i.e., $\hat{a}_S^{j,k} \geq \hat{a}_1^{j,k}$. Note that $D$ is the largest worst-case end-to-end delay bound for all flows traversing the network, we have $\hat{f}_i^{j,k} - \hat{a}_1^{j,k} \leq D$. From the definition of error term $\Psi_i$, we have $\tilde{\nu}_i^{j,k} - \hat{a}_1^{j,k} \leq D - \Psi_i \leq D$. Therefore, $\tilde{\nu}_i^{j,k} - \hat{a}_S^{j,k} \leq D$, or

$$\hat{a}_S^{j,k} \geq \tilde{\nu}_i^{j,k} - D. \tag{8}$$

Combining (7) and (8), we see that

$$\hat{a}_S^{j,k} - \hat{a}_1^{n,l} > D. \tag{9}$$

From this we know that packet $p^{n,l}$ has stayed in the network longer than $D$ when packet $p^{j,k}$ arrives at the scheduler at time $\hat{a}_S^{j,k}$. This contradicts the fact that $D$ is the largest worst-case end-to-end delay bound for all flows including flow $n$. Therefore we establish $\tilde{\nu}_i^{j,k} - \tilde{\nu}_i^{n,l} \leq 2D$ and the theorem follows. ■

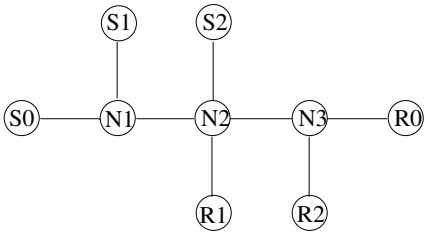


Fig. 5. Simulated network topology.

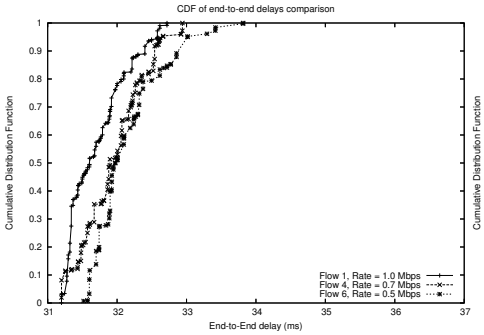


Fig. 6. CDF of end-to-end packet delays in FIFO network.

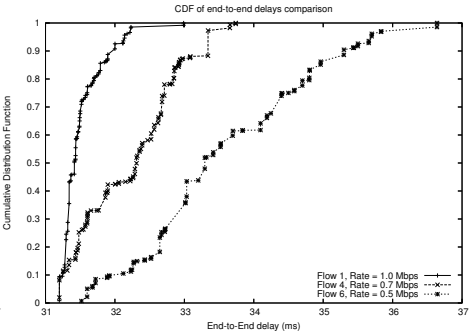


Fig. 7. CDF of end-to-end packet delays in CSVC network.

### C. Scheduling Complexity of BCSQ Scheduler

BCSQ has runtime scheduling complexity comparable to the bin-based event scheduling algorithm *Calendar Queue* [5] and the *stateful* packet scheduler Bin Sort Fair Queueing (BSFQ) [6]. In the following we analyze the complexity of the enqueue and dequeue operations of BCSQ respectively. Consider a BCSQ scheduler $i$ with $M$ bins. When a packet $p^{j,k}$ arrives at the scheduler, the scheduler checks if the virtual finish time of the packet falls in one of the current bins. If this is the case, the packet will be placed at the tail of the corresponding queue. This can be done with one insert operation. Otherwise, one or multiple bins need to be rotated. The rotation of bins can be done by simply updating the bin pointers instead of physically copying queues [14]. Therefore, in the worst case, a constant number of pointer operations $O(M)$ are needed to enqueue a new packet.

Now let us consider the dequeue operation of BCSQ. If the current queue is not empty, the scheduler can directly schedule the next packet in the queue after finishing serving the current packet, which only needs one delete operation. (Note that $currentqueue$ may be updated after a new packet arrives.) Otherwise, the scheduler needs to search the next non-empty queue, which, in the worst case, may take a constant number of $O(M)$ comparison operations.

In summary, the *worst-case* scheduling complexity of a BCSQ scheduler with $M$ bins is a constant $O(M)$. It only depends on the number of bins maintained by the scheduler, and is independent of the number of packets (flows) in the scheduler. However, similar to the observation made about calendar queues, the *amortized* per packet scheduling complexity of BCSQ is O(1) [5]. This can be understood by noting that incoming packets may occupy most of the bins or only a few bins in a scheduler. In the former case, it is likely that the scheduler will locate the next non-empty bin after examining a small number of empty bins. In the latter case, the scheduler may need to examine a relatively large number of empty bins before identifying the next non-empty bin. However, after the scheduler encounters a non-empty bin, it will serve that bin for a relatively long time before beginning to search the next non-empty bin again. This is because a large number of packets fall in the queues of a few bins. In either of these two cases, the average per-packet scheduling complexity of a BCSQ scheduler is only $O(1)$.

## IV. Simulation Studies

In this section we perform simulation studies to illustrate the efficacy of BCSQ, and to compare the performance of BCSQ with that of FIFO and the core stateless virtual clock scheduler CSVC (Section II). We also study the effects of time interval length $\iota$ on the performance of a BCSQ network.

Figure 5 depicts the network topology used in the simulation studies. All links have a capacity of $10$ Mbps and propagation delay of $10$ ms. Nodes $S0$, $S1$, and $S2$ are edge routers; they shape a flow's traffic according to the reservation rate of the flow. To make the comparisons fair for the three types of packet schedulers, edge traffic shaping is conducted in all the simulation studies. Moreover, all BCSQ schedulers in a network of BCSQ schedulers use the same bin time interval length $\iota$ in each specific simulation study. In reality, BCSQ schedulers in a BCSQ network may use different bin time interval lengths.

For the simulation results that we will report below, six CBR flows are originated from node $S0$ and destined to node $R0$. The average sending rates of the flows $1-6$ are $1$ Mbps, $0.9$ Mbps, $0.8$ Mbps, $0.7$ Mbps, $0.6$ Mbps, and $0.5$ Mbps, respectively. There are also six Exponential On/Off (Expoo) flows from $Si$ to $Ri$, for $i = 1, 2$. All packets are of a fixed size of $210$ bytes. The configuration of the Expoo flows is such that the average network utilization is $90\%$ (i.e., the average utilization of the bottleneck links is $90\%$). The reservation rate of a flow equals its average rate (for BCSQ and CSVC). For the packets of the CBR flows, we measured the delays between $N1$ and $R0$, and refer to them as the end-to-end delays of the packets. We have conducted simulation studies with different configurations by varying the number of CBR flows, Expoo flows, and network utilization. Similar observations were obtained.

Figs. 6 and 7 present the empirical cumulative distribution function (CDF) of the end-to-end delays experienced by packets of three CBR flows $1, 4, 6$ in the networks of FIFO and CSVC schedulers, respectively. As expected, flows with different sending rates experienced *similar* end-to-end packet

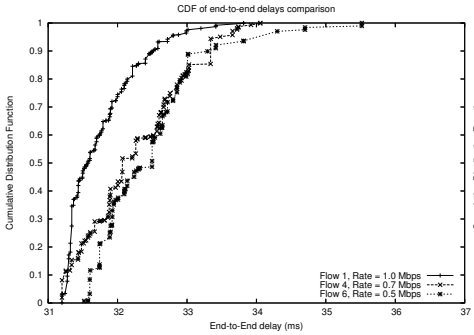


Fig. 8. CDF of end-to-end packet delays in BCSQ network ($\iota = 0.01$s).

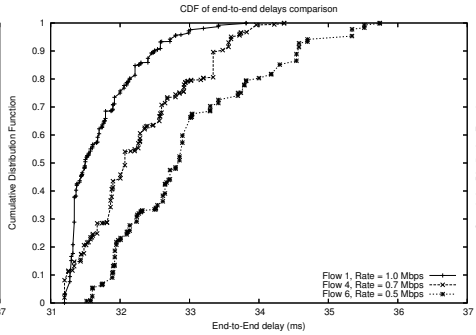


Fig. 9. CDF of end-to-end packet delays in BCSQ network ($\iota = 0.005$s).

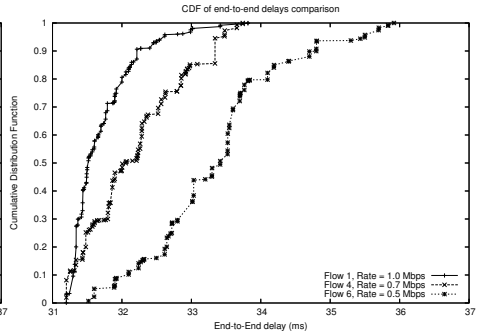


Fig. 10. CDF of end-to-end packet delays in BCSQ network ($\iota = 0.0025$s).

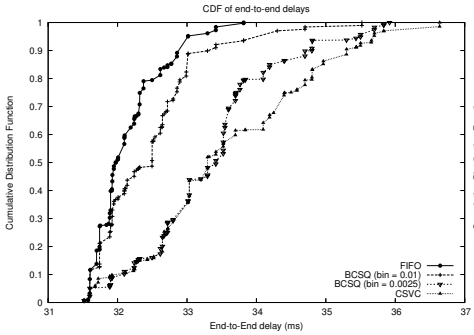


Fig. 11. CDF of end-to-end packet delays of flow 6 (rate = 0.5 Mbps).

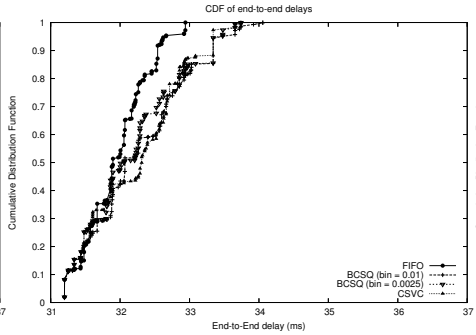


Fig. 12. CDF of end-to-end packet delays of flow 4 (rate = 0.7 Mbps).

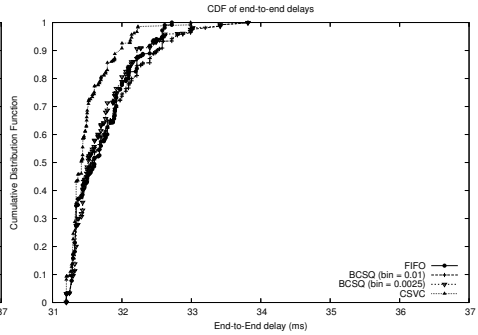


Fig. 13. CDF of end-to-end packet delays of flow 1 (rate = 1 Mbps).

delays in the FIFO network, given that a FIFO scheduler cannot differentiate flows. (Note that the CDF curves of the three flows are close to each other in Fig. 6.) In contrast, from Fig 7 we see that in the CSVC network, flows with higher reservation rates (such as flow 1) received better services in terms of end-to-end packet delays than flows with lower reservation rates (such as flow 6). This is because CSVC can differentiate the forwarding of packets based on the reservation rates of the flows they belong to (the reservation rate of a flow is carried in the packet header).

Figs. 8, 9, and 10 show the empirical CDF of end-to-end delays experienced by packets of the same three CBR flows in the BCSQ networks, with the bin time interval $\iota = 0.01$ s, $0.005$ s, and $0.0025$ s, respectively. Fig. 8 shows that when the bin time interval is large, BCSQ performs similarly as FIFO. This is because when the bin time interval is large, the majority of packets will fall into a small number of bins, and packets in the same bin are served in a FIFO manner. As an extreme case, if the bin time interval is sufficiently large so that all packets fall into the same bin, BCSQ will essentially degenerate into a FIFO scheduler. As we decrease the bin time interval gradually (Figs. 9 and 10), we see that BCSQ starts to better differentiate flows with different reservation rates at the cost of more bins being occupied, and consequently greater complexity. Like CSVC, flows with larger reservation rates receive better services in terms of end-to-end packet delays. From this set of simulation studies we conclude that, by controlling the bin time intervals of BCSQ schedulers, we can achieve different end-to-end per-flow service guarantees, albeit with different levels of runtime scheduling complexity.

From the above discussions we see that flows with different reservation rates will observe similar end-to-end service guarantees when we are not able to effectively differentiate them, for example, in a network of BCSQ with large bin time intervals or in a FIFO network. In this case flows with smaller reservation rates "steal" network resources such as bandwidth that should have been "allocated" to flows with larger reservation rates. As the ability to differentiate flows increases, flows with different reservation rates start to receive distinct service guarantees. These observations are confirmed in Figs. 11, 12, and 13, which show the empirical CDF of end-to-end delays experienced by packets in the different networks for flows 6, 4, and 1, respectively.

For flow 6 (Fig. 11), which has the lowest reservation rate ($0.5$ Mbps), the FIFO network provides the best end-to-end packet delay service compared to the BCSQ networks and the CSVC network. As we start to differentiate flows by deploying BCSQ (or CSVC) to provide service to flows based on their reservation rates, the service received by the flow 6 degrades compared to that in the FIFO network (see Fig. 11). It can no longer steal as much bandwidth as in the FIFO network. In contrast, flows with larger reservation rates receive better service when we are able to better differentiate flows. For example, flow 1 (Fig. 13), which has the largest

reservation rate, receives better service in a BCSQ network or a CSVC network compared that in the FIFO network. This is because, unlike the FIFO scheduler, BCSQ and CSVC schedulers can differentiate the forwarding of packets from flows with different reservation rates.

## V. Conclusion

In this paper we proposed a bin-based core stateless queueing (BCSQ) algorithm, which has constant runtime scheduling complexity that is independent of the number of packets (or flows) in the scheduler. BCSQ was developed in the virtual time reference system framework and can be deployed at core routers in VTRS. We presented the detailed design of BCSQ and derived the worst-case end-to-end packet delay bounds for flows in a network of BCSQ. We also studied the effects of the number of bins maintained by a BCSQ scheduler and the length of bin time intervals on the performance of BCSQ networks. In particular, we investigated the minimum number of bins that a scheduler needs to maintain in order to prevent packet overflows due to the limited time window the bins can collectively represent. Simulation studies were also conducted to illustrate the efficacy and performance of BCSQ. Through simulation studies we showed that, by controlling the length of time intervals the bins represent, BCSQ can achieve various performance and complexity trade-offs. When the bin time intervals are sufficiently long, all incoming packets will fall in a single bin and BCSQ degenerates into a FIFO scheduler. On the other hand, as we gradually decrease the length of time intervals, BCSQ is able to provide improved per-flow QoS guarantees, albeit with greater scheduling complexity.

## References


[1] Y. Bernet, J. Binder, S. Blake, M. Carlson, B. E. Carpenter, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, and W. Weiss. A framework for differentiated services. Internet Draft, February 1999. Work in Progress.

[2] S. Bhatnagar and B. Nath. Distributed admission control to support guaranteed services in core-stateless network. In *Proc. IEEE INFOCOM*, Sun Francisco, CA, April 2003.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, December 1998.

[4] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: An overview. RFC 1633, June 1994.

[5] R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10), October 1988.

[6] S. Cheung and C. Pencea. BSFQ: Bin sort fair queueing. In *Proc. IEEE INFOCOM*, New York, NY, June 2002.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. ACM SIGCOMM*, pages 1–12, Austin, TX, September 1989.

[8] Z. Duan, Z.-L. Zhang, and Y. T. Hou. Foundamental trade-offs in aggregate packet scheduling. *IEEE Transactions on Parallel and Distributed Systems*, December 2005.

[9] Z. Duan, Z.-L. Zhang, Y. T. Hou, and L. Gao. A core stateless bandwidth broker architecture for scalable support of guaranteed services. *IEEE Transactions on Parallel and Distributed Systems*, January 2004.

[10] J. Kaur and H. Vin. Core-stateless guaranteed rate scheduling algorithms. In *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.

[11] J. Kaur and H. Vin. Core-stateless guaranteed throughput networks. In *Proc. IEEE INFOCOM*, Francisco, CA, April 2003.

[12] J. Kaur and H. Vin. Providing deterministic end-to-end fairness guarantees in core-stateless networks. In *Proc. IEEE/IFIP Seventh International Workshop on Quality of Service (IWQoS '03)*, Monterey, CA, June 2003.

[13] C. Li and E. Knightly. Coordinated network scheduling: A framework for end-to-end services. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Osaka, Japan, November 2000.

[14] J. Liebeherr and D. E. Wrege. A versatile packet multiplexer for quality-of-service networks. In *Proc. 4th International Symposium on High Performance Distributed Computing (HPDC-4)*, pages 148–155, August 1995.

[15] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In *Proc. ACM SIGCOMM*, Boston, MA, September 1999.

[16] I. Stoica, H. Zhang, S. Shenker, R. Yavatkar, D. Stephens, A. Malis, Y. Bernet, Z. Wang, F. Baker, J. Wroclawski, C. Song, and R. Wilder. Per hop behaviors based on dynamic packet states. Internet Draft, February 1999. Work in Progress.

[17] A. Terzis, L. Wang, J. Ogawa, and L. Zhang. A two-tier resource management model for the internet. In *Global Internet 99*, December 1999.

[18] J. Xu and R. Lipton. On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms. In *Proc. of ACM SIGCOMM 2002*, August 2002.

[19] Z.-L. Zhang, Z. Duan, L. Gao, and Y. T. Hou. Decoupling QoS control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In *Proc. ACM SIGCOMM*, Sweden, August 2000.

[20] Z.-L. Zhang, Z. Duan, and Y. T. Hou. Virtual time reference system: A unifying scheduling framework for scalable support of guaranteed services. *IEEE Journal on Selected Areas in Communication,* Special Issue on Internet QoS, December 2000.