Packet Scheduling in a Combined Input and Output Queued Switch Using Virtual Time Reference System

Yiwei Thomas Hou* Zhi-Li Zhang† Takeo Hamada‡

Abstract

There is an increasing demand for Internet core nodes to have (1) quality-of-service (QoS) guarantees and (2) higher switching capacity. The combined input and output queued (CIOQ) switching has been shown to be a promising solution to meet such demand. However, many of the proposed scheduling algorithms for a CIOQ switch cannot be practically implemented due to problems from scalability and complexity. This paper shows how the virtual time reference system (VTRS) proposed in [15] can be applied to packet scheduling in a CIOQ switch. The VTRS is a unifying scheduling framework to provide scalable support for guaranteed services. In the context of packet scheduling for a CIOQ switch, we show that the use of VTRS can eliminates both the scalability and complexity problems associated with many of the scheduling algorithms proposed in the literature. More important, we show that in term of of providing end-to-end guaranteed service, packet scheduling using VTRS for a CIOQ switch has the same performance as an output queued (OQ) switch employing weighted fair queueing (WFQ) scheduler.

Key Words: Packet switching; Scheduling; QoS; Guaranteed service; Weighted fair queueing; Scalability

1 Introduction

The Internet backbone is facing two problems simultaneously: (1) there is a need to introduce guaranteed QoS, and (2) there is a need for faster switching infrastructure. The first problem can be solved by building switches using the so-called output-queueing,[1] which is employed by many commercial switches and routers today. This approach is known to achieve the throughput of the switch to 100%. Furthermore, powerful scheduling algorithms (e.g., WFQ [5, 9]) can be placed at the output port and thus provide QoS guarantee. But output queueing for an $N \times N$ switch requires the switching fabric and memory to run N times as fast as the line rate. This is impractical to design high speed switches (e.g., terabit switch) since memories with sufficient bandwidth are simply not available at such high speed.

To build faster switches, an input-queued (IQ) switch architecture can be employed since the fabric and memory of an IQ switch need only run as fast as the line rate. Furthermore, it has been shown that by using a scheme known as *virtual output queueing* (VOQ)[2], it is possible to eliminate entirely the so-called *head-of-line* (HOL) blocking problem[3] associated with an IQ switch. However, it remains to be seen how an IQ switch without any speedup can guarantee QoS.

Recently, it has been shown that it is possible to use a *combined input and output port queued* (CIOQ) switch with a small speedup (e.g., 2–4) to provide guaranteed QoS [3, 11]. Under such architecture, buffers are employed at both the input ports and output ports and the switch can remove up to S $(1 \leq S \leq N)$ packets from each input and deliver up to S packets to each output within a time slot, where a time slot is the time between packet arrival at input ports. It has been shown in [3, 11] that with a small speedup (e.g., 2–4), a CIOQ switch can behave *identically* to an OQ switch for *all* types of traffic. Here, "behave identically" means that, when the same inputs are applied to both the OQ switch and to the CIOQ switch, the corresponding output processes from the two switches are completely indistinguishable.

Although the algorithms presented in [3, 11] theoretically enable a CIOQ switch with a small speedup to mimic an OQ switch with WFQ scheduling algorithm, in practice, such algorithms may not be implementable for the following two problems: (1) it is not scalable (and thus not feasible) to maintain per-flow QoS state in a high speed CIOQ switch to mimic an OQ switch with WFQ scheduler, and (2) even if it *were* feasible to maintain per-flow state at the CIOQ switch, the calculation of departure time for scheduling in a CIOQ switch in [3, 11] requires complex communication among various input and output ports, which is simply not implementable for a switch operating at very high speed. Therefore, there is a need to design an implementation-friendly scheduling algorithm for a CIOQ to guarantee QoS.

This paper shows how to use a novel *virtual time reference system* (VTRS) proposed by Zhang et. al [15] to perform packet scheduling in a CIOQ switch and to achieve *end-to-end* QoS guarantee.

The VTRS extends the work of Stoica and Zhang [14] and is a a *unifying* scheduling framework to provide scalable support for guaranteed services. In the same way that the WFQ reference system relates to the Integrated Services (IntServ) architecture [2, 4], the VTRS is designed as a *conceptual* framework upon which guaranteed services [12] can be implemented in a scalable man-

*Y. T. Hou is with Fujitsu Laboratories of America, Sunnyvale, CA, USA.

†Z.-L. Zhang is with University of Minnesota, Minneapolis, MN, USA.

‡T. Hamada is with Fujitsu Laboratories of America, Sunnyvale, CA, USA.

[1] When we refer to output-queueing in this paper, we include designs that employ centralized shared memory.

[2] VOQ refers that each input maintains a separate queue for each output [8].

[3] HOL blocking refers that if each input port maintains a single FIFO, the throughput of an IQ switch is limited to just 58.6% (under uniform traffic) [7].

ner using the Differentiated Services (DiffServ) paradigm [1]. More specifically, the VTRS provides a unifying framework to characterize, in terms of their abilities to provide delay and bandwidth guarantees, both the *per-hop behaviors* of core routers and the *end-to-end properties* of their concatenation. The key construct in the VTRS is the notion of *packet virtual time stamps,* which, as part of the packet state, are referenced and updated as packets traverse each core switch. A key property of packet virtual time stamps is that they can be computed using solely the packet state carried by packets (plus a couple of fixed parameters associated with core routers). In this sense, the VTRS is *core stateless,* as no per-flow state is needed at core routers for computing packet virtual time stamps.

This paper shows that the VTRS can be applied to packet scheduling in a CIOQ switch and to resolve *both* the scalability problem (associated with maintaining per-flow QoS state information at a switch) and the complex inter-port communication problem (encountered in calculating and updating departure time for each packet) associated with an CIOQ switch in [3, 11]. More specifically, the *virtual finish time* associated with each incoming packet to a CIOQ switch can be used for scheduling in a CIOQ switch to mimic a *core stateless virtual clock* ($C_{\mathcal{S}}$VC). The novelty and power of using virtual finish time associated with each packet under the VTRS is: virtual finish time can be calculated *directly* from the packet virtual time stamps carried in the packet and thus eliminates the needs of maintaining the per-flow QoS state information at a switch as well as the complex inter-port communications required to calculate/update packet departure time used in [3, 11]. More important, since the $C_{\mathcal{S}}$VC has the same per hop delay performance as a WFQ, in terms of end-to-end delay guarantee, a CIOQ switch using virtual finish time for scheduling provides the same performance as an OQ switch with WFQ scheduler.

This rest of this paper is organized as follows. In Section 2, we outline the architecture of a CIOQ switch as well as some proposed packet scheduling algorithms. We show the implementation problems associated with scheduling in a CIOQ switch in order to mimic an OQ switch under WFQ scheduler. Section 3 presents the framework of VTRS. In Section 4, we show how to apply the VTRS for packet scheduling in a CIOQ switch and provide end-to-end QoS guarantee. Section 5 concludes this paper.

2 Architecture of a CIOQ Switch

2.1 Basic Architecture

Consider a single stage, $N \times N$ CIOQ switch. Under VOQ, each input maintains a separate queue for packets destined for each output (see Fig. 1(a) for an example 3×3 crossbar CIOQ switch). For simplicity, we assume all input and output buffers have infinite capacity. Although packets arriving to the switch may have variable length, we will assume that they are treated internally as fixed length packets (or "cell"). This is common practice in high performance switches; variable length packets are segmented into fixed length cells at the input ports, transferred across the switch fabric as cells, and reassembled back into packets again at the output ports. For the ease of exposition, we assume all packets have

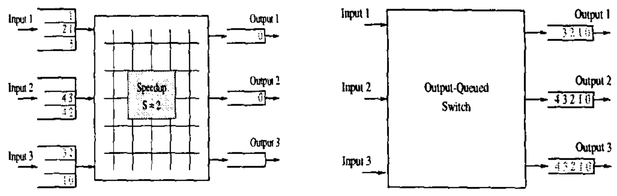

Figure 1: An example 3×3 crossbar CIOQ switch with $S = 2$ (a) and its reference OQ switch (b).

the same fixed length and we take the time between such fixed-length packet as the basic time unit.

A scheduling algorithm selects a matching between inputs and outputs in such a way that each non-empty input is matched with at most one output and, conversely, each output is matched with at most one input. The matching is used to configure the switch before packets are transferred from the input side to the output side. A CIOQ switch with a speed up of S is able to make S such transfers each time slot.

2.2 Mimicking FIFO OQ Switch

In an OQ switch (see Fig. 1(b)), arriving packets are immediately forwarded to their corresponding outputs. This (1) ensures that outputs never idle so long as there is a packet destined for them in the system, and (2) allows the departure of packets to be scheduled (e.g., WFQ) to meet delay constraints. In [11], Prabhakar and McKeown presented a novel scheduling algorithm called the *Most Urgent Cell First Algorithm* (MUCFA) for a CIOQ switch to mimic a FIFO OQ switch. They show that with a speedup of $S = 4$, an $N \times N$ switch operating under MUCFA can behave identically to a FIFO OQ switch, regardless of input traffic patterns and for any switch size N.

A key aspect of MUCFA is the concept of the "urgency of a packet", which is defined in reference to a *shadow* OQ switch (see Figure 1). With respect to the OQ switch (Figure 1(b)), each arriving cell to this switch is stamped with a number, which is its "urgency value" at *that* time. This number indicates the time from the *present* that it will depart from the switch. At each successive time slot, the urgency value is decremented by one. When the value reaches zero, the cell will depart at the end of this time slot. If two packets a and b arriving at two different inputs at the beginning of the same time slot and both destine for the same output port, then the urgency of packet a, say u_a, is less than the urgency of packet b, say u_b, if and only if the number of the input port at which packet a arrives is less than the number of the input port at which packet b arrives. That is, the OQ switch is assumed to transfer packets from inputs to outputs in a round robin fashion starting with the smallest numbered input first.

Now consider the CIOQ switch (Figure 1(a)). By assumption, the same input is applied to it and to the OQ switch. Thus for every packet p in the CIOQ switch, there is an exact copy in the OQ switch, which is referred to as the *clone* of p. Therefore, packet p arrives at input i at time T and is destined for output j. Since the speedup $1 \leq S \leq N$, packet p may not be forwarded to the output buffer j at the end of time slot T. Note that packet p *may not be required* at output j for some time, because its clone in the OQ switch may still be some distance from the HOL. Therefore, the urgency is an indication of how much time there is before packet p is needed at its output if the CIOQ switch is to mimic the behavior of the OQ switch. The following is the formally definition for urgency.

Definition 1 *The urgency of a packet in a CIOQ switch at any time is the distance its clone is from the head of the output buffer in the corresponding reference OQ switch.* □

The packets in any output buffer of the CIOQ switch are arranged in increasing order of urgencies, with the most urgent packet at the head. Once packet p is forwarded to its output in the CIOQ switch, its position is determined by its urgency.

Algorithm 1 (MUCFA)

1. At the beginning of each phase[4], each output tries to obtain its most urgent packet from the input.
2. If more than one output request the same input, then the input will grant to that output whose packet has the smallest urgency value. If there is a tie between two or more outputs, then the output with the smallest port number wins.
3. Output that lose such contention at an input will try to obtain their next most urgent packet from another (unmatched) input port.
4. When no more matching of inputs and outputs is possible, packets are transferred and MUCFA goes to the next phases (Step 1). □

Theorem 1 *An $N \times N$ CIOQ switch operating under MUCFA and speedup $S \geq 4$, can behave identically to a FIFO OQ switch, regardless of input traffic patterns and for arbitrary switch size N.* □

The proof of Theorem 1 can be found in [11].

2.3 From FIFO to WFQ

The MUCFA can be used to mimic an OQ switch employing a wider range of output scheduling policies than just FIFO. Essentially, the extension to these non-FIFO scheduling policies involves very little change to the basic structure of MUCFA.

The MUCFA can be extended to the so-called *monotone* scheduling policy, which is defined as follows.

Definition 2 *An output scheduling policy is said to be monotone if, once it has been determined, the relative departure order of any two packets p and q does not change over time.* □

[4]For a CIOQ switch with speedup S, a time slot is said to be divided into S equal phases. During each phase ϕ_i, $1 \leq i \leq S$, the switch can remove at most one packet from each input port and can transfer at most one packet to each output port. It is assumed that packets arriving at the switch input ports will do so at the beginning of phase ϕ_1, while departures from the switch output ports take place at the end of phase ϕ_S.

A simple way of visualizing this class of output scheduling policies is to imagine a single "push-in" queue at the output, where an arriving packet may be pushed into any location but packets may depart only from front. Note that newly arriving packets may increase the absolute departure time of an existing packet, but cannot change the position of the existing packet relative to another. The importance of the class of monotone policies is that it includes several policies that are commonly used to provide QoS guarantees (e.g., WFQ).

The following definition of *expected urgency* (EU)[5] extends the definition for urgency and sets the stage to extend MUCFA for monotone output scheduling policies.

Definition 3 *The expected urgency, $EU_p(t)$, of a packet p at an time slot t is the time from the present that it would depart from the switch if no new packets arrive to the switch after time t.* □

When the output scheduling policy is FIFO, the $EU_p(t)$ of packet p is the same as the its urgency. For FIFO output scheduling policies $EU_p(t)$ decreases exactly by one every time slot. This need not be the case under a general monotone output scheduling policy since new packets may be pushed into the output queue and take precedence over an existing packet, causing its EU to *increase.*

Let MUCFA-E be the algorithm that during any phase of time slot t schedules the transfer of packets from inputs to outputs in exactly the same manner as MUCFA, except for basing its scheduling decisions on a packet's expected urgency EU instead of its urgency U. It have been shown that MUCFA-E may be used in a CIOQ switch with a speed of $S \geq 4$ to behave identically to an OQ switch employing any monotone scheduling policy [11].

2.4 Implementation

There are two components in implementing the MUCFA for CIOQ switch: (1) The process of matching inputs and outputs for transferring packets, and (2) Determining the urgency of an arriving packet. Regarding the first problem, it has been shown in [11] that for an $N \times N$ CIOQ switch employing MUCFA and an internal speedup of at least 4, the number of iterations required to match inputs and outputs in each phase is never more than N.

As for the second problem, it is not surprising that the difficulty of inferring the urgency depends on the output scheduling policy of the reference OQ switch. For the simple FIFO (and strict priority scheduling), the implementation is straightforward and can be practically implemented. But the difficulty quickly changes when it comes to other scheduling policy such as WFQ, which we elaborate as follows.

[5]Actually, the term "expected departure time" instead of "expected urgency" is used in [11]. In order to avoid confusion with our own definition of "expected departure time" in this paper, we opt to use the term "expected urgency."

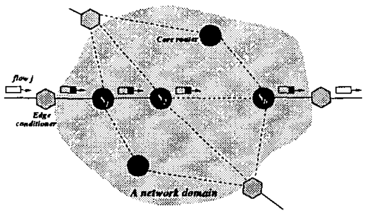

Figure 2: A conceptual network model for VTRS.

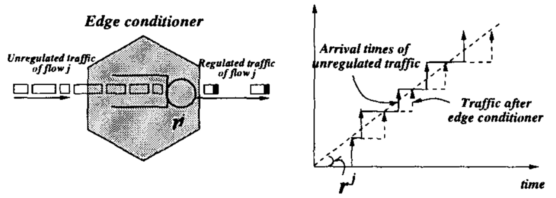
Edge conditioner
Unregulated traffic of flow j
Regulated traffic of flow j
r^j
Arrival times of unregulated traffic
Traffic after edge conditioner
r^j
time

Figure 3: Edge conditioning (or shaping) and its effect in the VTRS.

2.4.1 An Open Problem

The implementation of MUCFA-E for a CIOQ switch to mimic an OQ switch under WFQ scheduler face the the following two problems: (1) it is not scalable (and thus may not be feasible) to maintain per-flow QoS state at a high speed CIOQ switch, and (2) even if it *were* feasible to maintain per-flow state at the switch, the calculation of departure time for scheduling in a CIOQ switch in [3, 11] requires complex communication among various input and output ports, which is simply not implementable for a switch operating at very high speed. Therefore, there is a need to design an implementation-friendly scheduling algorithm for a CIOQ to guarantee QoS.

3 Virtual Time Reference System

3.1 Basic Architecture

The VTRS is defined and implemented within a *single* administrative domain. Conceptually, the VTRS consists of three logical components (see Figs. 2, 3 and 4): *packet state* carried by packets, *edge traffic conditioning* at the network edge, and *per-hop virtual time reference/update mechanism* at core switches or routers (e.g., CIOQ switch). The packet state carried by a packet contains three types of information: (1) QoS reservation information of the flow the packet belongs to (e.g., the reserved rate of the flow); (2) a virtual time stamp of the packet; and (3) a virtual time adjustment term.

We summarize the important notation used in the paper as follows.

General Notation
$p^{j,k}$: the kth packet of flow j
$L^{j,k}$: packet length of $p^{j,k}$
$L^{j,max}$: maximum packet length of flow j
$L^{*,max}$: maximum packet length of all flows at a node
r^j: reserved rate of flow j
h: number of hops (nodes) along the path of flow j

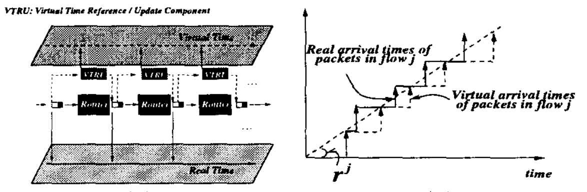
VTRU: Virtual Time Reference / Update Component
Virtual Time
VTRU
VTRU
VTRU
Router
Router
Router
Real Time
(a)
Real arrival times of packets in flow j
Virtual arrival times of packets in flow j
r^j
time
(b)

Figure 4: A schematic of VTRS: (a) Virtual time reference/update mechanism; and (b) Virtual traffic shaping.

Notation for the Ideal Per-Flow System
$a_i^{j,k}$: arrival time of packet $p^{j,k}$ at node i
$f_i^{j,k}$: finish time of packet $p^{j,k}$ at node i
$\Delta_i^{j,k}$: cumulative queueing delay packet $p^{j,k}$ experienced up to server i (inclusive)

Notation for the Virtual Time Reference System
$\tilde{\omega}_i^{j,k}$: virtual time stamp of packet $p^{j,k}$ at node i
$\tilde{\nu}_i^{j,k}$: virtual finish time of packet $p^{j,k}$ at node i
$\delta^{j,k}$: virtual time adjustment term for packet $p^{j,k}$: $\delta^{j,k} = \Delta_i^{j,k}/i$
$\tilde{d}_i^{j,k}$: virtual delay of packet $p^{j,k}$ at node i: $\tilde{d}_i^{j,k} = \tilde{\nu}_i^{j,k} - \tilde{\omega}_i^{j,k}$
$\hat{a}_i^{j,k}$: actual time packet $p^{j,k}$ arrives at node i
$\hat{f}_i^{j,k}$: actual time packet $p^{j,k}$ departs from node i
Ψ_i: error term of scheduling blackbox at node i
$\pi_{i,i+1}$: propagation delay from the ith node to the $(i+1)$th node

Edge traffic conditioning ensures that traffic of a flow will never be injected into the network core at a rate exceeding its reserved rate (see Fig. 3). Formally, for a flow j with a reserved rate r^j, the inter-arrival time of two consecutive packets of the flow is such that

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

As a packet traverses each core router along the path of its flow, a virtual time stamp is "attached" to the packet. This virtual time stamp represents the arrival time of the packet at the core router *in the virtual time,* and thus it is also referred to as the *virtual arrival time* of the packet at the core router. The virtual time stamps associated with packets of a flow satisfy an important property, which we refer to as the *virtual spacing property* as follows:

$$\tilde{\omega}^{j,k+1} - \tilde{\omega}^{j,k} \geq \frac{L^{j,k+1}}{r^j} \tag{2}$$

for all k. Comparing (2) with (1), we see that *with respect to the virtual time,* the inter-arrival time spacing is preserved at a core router. Another key property of packet virtual time stamps is that *at a core router the virtual arrival time of a packet always lags behind its real arrival time.* This property (referred to as the *reality check condition*) is important in deriving end-to-end delay bound

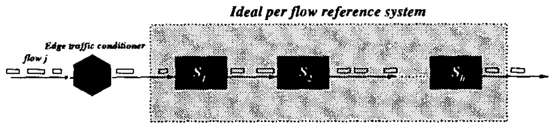

Figure 5: An ideal per-flow system.

experienced by packets of a flow across the network core. The per-hop virtual time reference/update mechanism at a core router is designed in such a manner so as to ensure that these properties of the packet virtual time stamps are satisfied at the entry point and/or exit point of the core router (see the illustration in Fig. 4).

Conceptually, for each packet traversing a core router, a *virtual finish time* is computed and assigned to it. This virtual finish time is derived from its virtual time stamp and other packet state information. Intuitively, it represents the time the packet finishes its service in an *ideal per-flow reference system,* where the flow to which the packet belongs to is the only flow serviced by the system. *The per-hop behavior of a core router is defined in terms of an upper bound on the difference between the actual departure time and virtual finish time of a packet traversing the core router.* This upper bound is referred to as the *error term* of the core router. Therefore, the scheduling mechanism of the core router can be abstracted into a *scheduling blackbox* characterized by an error term. This simple abstraction enables us to derive *end-to-end delay bounds* for flows traversing an arbitrary concatenation of such scheduling blackboxes.

3.2 An Ideal Per-flow VTRS

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

Figure 5 illustrates an ideal per-flow system, where a regulated flow is serviced by a dedicated channel. The dedicated channel consists of a series of servers in tandem. Packets of a flow j are serviced *in order* from server 1 to server h. For simplicity of exposition here, we assume that in this ideal per-flow system the propagation delay from one server to the next server is zero. (We will take into account of propagation delay in the next subsection.) For the purpose of this paper, we will only consider the so-called *rate-based servers.*[6] A rate-based server has a service capacity equal to the reserved rate r^j of flow j. Hence a rate-based server takes $L^{j,k}/r^j$ amount of time to process packet $p^{j,k}$ of flow j.

[6] A more general study including *delay-based servers* can be found in [15].

3.2.1 End-to-end Delay

Let $a_1^{j,k}$ be the arrival time[7] of packet $p^{j,k}$ of flow j at the first server of the ideal per-flow system. Then the edge spacing condition holds, namely, $a_1^{j,k+1} - a_1^{j,k} \geq L^{j,k+1}/r^j$ for $k = 1, 2, \ldots$. For $i = 1, 2, \ldots, h$, let $a_i^{j,k}$ denote the time packet $p^{j,k}$ arrives at server $\mathcal{S}_i$, and $f_i^{j,k}$ the time it leaves server i. In the ideal per-flow system, it is not hard to see that the following recursive relationships among $a_i^{j,k}$'s and $f_i^{j,k}$'s hold. For any $k = 1, 2, \ldots$, $a_i^{j,k} = f_{i-1}^{j,k}$ $i = 2, \ldots, h$, and

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

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

For $i = 1, 2, \ldots, h$, let $\Delta_i^{j,k}$ denote the cumulative queueing delay experienced by packet $p^{j,k}$ up to server i (inclusive). Formally, $\Delta_i^{j,k} = f_i^{j,k} - (a_1^{j,k} + i\frac{L^{j,k}}{r^j})$. We can derive an important recursive relation, $\Delta_i^{j,k}$ to $\Delta_i^{j,k-1}$ and the arrival times of packets $p^{j,k-1}$ and $p^{j,k}$ at the first-hop server as follows [15]: For any packet $p^{j,k}$, $k = 1, \ldots$, and $i = 1, 2, \ldots, h$, $\Delta_i^{j,1} = 0$, and

$$\Delta_i^{j,k} = \max\{0, \Delta_i^{j,k-1} + i\frac{L^{j,k-1} - L^{j,k}}{r^j} + a_1^{j,k-1} - a_1^{j,k} + \frac{L^{j,k}}{r^j}\}. \tag{3}$$

The importance of this relation lies in the fact that for each $p^{j,k}$, $\Delta_h^{j,k}$ can be calculated (recursively) *at the network edge.*

Therefore, the end-to-end delay of packet $p^{j,k}$ in the ideal per-flow system is $f_h^{j,k} - a_1^{j,k} = \Delta_h^{j,k} + h\frac{L^{j,k}}{r^j}$. In particular, it can be shown that $\Delta_h^{j,k} + hL^{j,k}/r^j \leq hL^{j,max}/r^j$. Thus, $f_h^{j,k} - a_1^{j,k} \leq h\frac{L^{j,max}}{r^j}$.

3.2.2 Packet Virtual Time Stamps

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

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

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

Virtual Spacing: for $k = 1, 2, \ldots$, $\tilde{\omega}_i^{j,k+1} - \tilde{\omega}_i^{j,k} \geq \frac{L^{j,k+1}}{r^j}$.

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

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

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

In the following we provide a definition of packet virtual time stamps for the ideal per-flow system, and show that it satisfies all the four properties listed above.

Consider the ideal per-flow system shown in Fig. 5. For each packet $p^{j,k}$, define $\delta^{j,k} = \Delta_h^{j,k}/h$. For $i = 1, 2, \ldots, h$, the *virtual delay* $\tilde{d}_i^{j,k}$ associated with packet $p^{j,k}$ at server $\mathcal{S}_i$ is computed from the packet state information using the following formula:

$$\tilde{d}_i^{j,k} = \frac{L^{j,k}}{r^j} + \delta^{j,k}.$$

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

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

$$\tilde{\omega}_i^{j,k} = \tilde{\nu}_{i-1}^{j,k} = \tilde{\omega}_{i-1}^{j,k} + \tilde{d}_{i-1}^{j,k} \text{ and } \tilde{\nu}_i^{j,k} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k}. \quad (4)$$

From the above definition, it is clear that the core stateless property holds trivially. It can be shown that the other three properties are also satisfied [15].

3.3 VTRS and Packet Scheduling in Real Network System

To extend the VTRS defined in the context of ideal per-flow system to a network system where each core router is shared by multiple flows, we introduce a key notion called the *error term* of a core router (or rather, of its scheduling mechanism).

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

$$\hat{a}_1^{j,k+1} - \hat{a}_1^{j,k} \geq \frac{L^{j,k+1}}{r^j} \quad (5)$$

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

As shown in Fig. 2, the path of flow j consists of h core routers, each of which employs certain scheduling mechanism to provide guaranteed service for flow j. For $i = 1, 2, \ldots, h$, we will refer to the scheduler at core router i as a scheduling *blackbox*, and denote it by $\mathcal{S}_i$. In the following, we will first characterize the *per-hop behavior* of the scheduling blackboxes, and then show how end-to-end delay bounds can be derived based on this characterization of their per-hop behavior.

3.3.1 Scheduling Blackbox

For a rate-based scheduling blackbox $\mathcal{S}_i$, packet $p^{j,k}$ of flow j is assigned a virtual delay $\tilde{d}_i^{j,k} = L^{j,k}/r^j + \delta^{j,k}$. For any flow j traversing the scheduling blackbox $\mathcal{S}_i$, let $\tilde{\omega}_i^{j,k}$ be the virtual time stamp associated with packet $p^{j,k}$ as it enters $\mathcal{S}_i$. We will provide a definition for $\tilde{\omega}_i^{j,k}$ shortly and establish its properties. At this point, we only assume that the *reality check condition* holds at $\mathcal{S}_i$, namely, $\hat{a}_i^{j,k} \leq \tilde{\omega}_i^{j,k}$, where $\hat{a}_i^{j,k}$ is the *actual* time that packet $p^{j,k}$ enters the scheduling blackbox $\mathcal{S}_i$. At $\mathcal{S}_i$, packet $p^{j,k}$ is assigned a virtual finish time $\tilde{\nu}_i^{j,k}$, where $\tilde{\nu}_i^{j,k} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k}$. Let $\hat{f}_i^{j,k}$ denote the *actual* time packet $p^{j,k}$ departs $\mathcal{S}_i^{j,k}$, i.e., $\hat{f}_i^{j,k}$ is the *real finish time* of $p^{j,k}$. We say that the scheduling blackbox $\mathcal{S}_i$ can *guarantee* packets of flow j their virtual delays with an *error term* Ψ_i, if for any k,

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

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

By using the packet virtual finish time as a reference point to quantify the real finish time of a packet at a core router, we are able to abstract and characterize the per-hop behavior of a core router via an error term. This error term captures the ability of the core router to provide guaranteed services to a flow.

3.3.2 End-to-End Delay Bounds

Consider the path of flow j shown in Figure 2. For $i = 1, 2, \ldots, h$, let Ψ_i be the error term associated with the scheduling blackbox $\mathcal{S}_i$. We now define $\tilde{\omega}_i^{j,k}$ and show that this definition satisfies the four requirements of packet virtual time stamps, namely, the *virtual spacing property*, the *reality check condition*, the *bounded delay property* and the *core stateless property*. Here in defining the reality check condition and bounded delay property, the quantities $a_i^{j,k}$ and $f_i^{j,k}$ defined in Section 3.2.2 are replaced by $\hat{a}_i^{j,k}$ and $\hat{f}_i^{j,k}$, which denote the *real arrival time* and *real finish time* of packet $p^{j,k}$ at $\mathcal{S}_i$, respectively.

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

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

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

From (5), the virtual spacing property is clearly met at the first-hop router. Furthermore, the reality check condition also holds trivially. Therefore, by the definition of Ψ_1, we have

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

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

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

By the definition of Ψ_i, we have

$$\hat{a}_{i+1}^{j,k} \leq \tilde{\nu}_i^{j,k} + \Psi_i + \pi_{i,i+1}. \tag{6}$$

In order to ensure that the reality check condition holds as packet $p^{j,k}$ enters the $(i+1)$th hop router $\mathcal{S}_{i+1}$, the relation (6) suggests that the virtual time stamp $\tilde{\omega}_{i+1}^{j,k}$ associated with packet $p^{j,k}$ at $\mathcal{S}_{i+1}$ should be defined as follows:

$$\tilde{\omega}_{i+1}^{j,k} = \tilde{\nu}_i^{j,k} + \Psi_i + \pi_{i,i+1} = \tilde{\omega}_i^{j,k} + \tilde{d}_i^{j,k} + \Psi_i + \pi_{i,i+1}. \tag{7}$$

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

Since Ψ_i's and $\pi_{i,i+1}$'s are *fixed* parameters associated with the core routers and the path of flow j, it is clear that the packet virtual time stamps defined using (7) are *core stateless*. Namely, they can be computed at each core router using only the packet state information carried by the packets (in addition to the two fixed parameters associated with the routers and the flow's path). Thus *no per-flow state needs to be maintained at these core routers.*

Since $\Psi_i + \pi_{i,i+1}$ is a constant independent of $p^{j,k}$, comparing the definition of $\tilde{\omega}_i^{j,k}$ in (7) and that in (4), it is easy to see that the virtual spacing property also holds at each core router $\mathcal{S}_i$. Furthermore, we have

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

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

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

This completes the construction of packet virtual time stamps for flow j.

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

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

This bound is analogous to those derived for fair-queueing/latency-rate-server based scheduling algorithms [6, 10, 13]. In particular, if $\Psi_i = L^{*,max}/C_i$, where $L^{*,max}$ is the maximum packet size permissible at the ith router and C_i is its service capacity, then the above inequality yields precisely the same delay bound as is obtained for a flow in a network of WFQ schedulers [5, 10] (or Virtual Clock (VC) schedulers [6]).

3.4 Core Stateless Virtual Clock Scheduling Algorithm

The notion of packet virtual time stamps can be used to design new *core stateless* scheduling algorithms. For the purpose of this paper, we only show a particular rate-based core stateless scheduling algorithm and establish its error terms using the properties of packet virtual time stamps.

A *core stateless virtual clock* ($C_{\mathcal{S}}$VC) scheduler $\mathcal{S}$ is a rate-based scheduler. It services packets in the order of their virtual finish time. For any packet $p^{j,k}$ traversing $\mathcal{S}$, let $\tilde{\omega}^{j,k}$ be the virtual time carried by $p^{j,k}$ as it enters $\mathcal{S}$, and $\tilde{d}^{j,k} = \frac{L^{j,k}}{r^j} + \delta^{j,k}$ be its virtual delay. Then the virtual finish time $\tilde{\nu}^{j,k}$ of $p^{j,k}$ is given by $\tilde{\omega}^{j,k} + \tilde{d}^{j,k}$. It can be shown that the $C_{\mathcal{S}}$VC scheduler can guarantee each flow j its reserved rate r^j with the minimum error term $\Psi_{C_{\mathcal{S}}VC} = L^{*,max}/C$, provided that an appropriate schedulability condition is met. This fact is stated formally in the following theorem, the proof of which can be found in [15].

Theorem 2 *Consider N flows traversing a $C_{\mathcal{S}}VC$ scheduler $\mathcal{S}$ such that the schedulability condition $\sum_{j=1}^{N} r^j \leq C$ is satisfied. Suppose that $\hat{a}^{j,k} \leq \tilde{\omega}^{j,k}$ for any packet $p^{j,k}$ of flow j, $j = 1, 2, \ldots, N$. Then*

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

In other words, $\Psi_{C_{\mathcal{S}}VC} = \frac{L^{*,max}}{C}$. □

3.5 Latency-Rate Servers and the VTRS

The VTRS framework does not exclude the use of *stateful* scheduling algorithms, namely, those scheduling algorithms that maintain per-flow state information in order to provide guaranteed services. To accommodate these stateful scheduling algorithms into the VTRS framework, it suffices to identify the error term incurred by these stateful scheduling algorithms. As an example to show how this can be done generally, we consider the class of scheduling algorithms introduced in [13]—the *latency-rate servers.* This class encompasses virtually all known fair-queueing algorithms and its variations. It can be shown that any latency-rate server with a latency Θ^j (with respect to flow j) has an error term such that $\Psi \leq \Theta^j$.

For several well-known scheduling algorithms studied in [13], it can actually be shown that $\Psi = \Theta^j - \frac{L^{j,max}}{r^j}$. In particular, Θ^j for PGPS/WFQ, VC, FFQ and SPFQ are all $\frac{L^{j,max}}{r^j} + \frac{L^{*,max}}{C}$ and each has the error term of $\frac{L^{*,max}}{C}$.

4 Using VTRS for Scheduling in a CIOQ Switch

In this section, we show how to apply the VTRS for packet scheduling in a CIOQ switch so as to resolve both the the scalability problem and complex inter-port communication problems encountered in [3, 11]. More important, we show that in terms of providing end-to-end guaranteed service [12], packet scheduling using VTRS in a CIOQ switch has the same performance as an OQ switch using WFQ scheduler.

4.1 Extension of MUCFA

Now let's revisit the MUCFA in Algorithm 1. Recall that the MUCFA is a scheduling algorithm used by a CIOQ switch to mimic an OQ switch with FIFO scheduling.

Under the MUCFA, the urgency of each packet, say U, is decremented by one at the end of each time slot. Denote T the current time slot for U, then using $U + T$ instead of U for scheduling in MUCFA would yield the same outcome as the original MUCFA because this merely adds the same current T for all packets in the switch.

Denote D as the *departure time* of the packet and let T_A and U_A be the *arrival time* and *urgency upon arrival* of the packet, respectively. In order words, U_A is the urgency of the packet upon arrival at time T_A. It is easy to see that the following relation holds:

$$D = T_A + U_A = T + U.$$

The above is true since (1) time T is incremented by 1 at the end of each time slot while urgency U is decremented by 1 at the end of each time slot, and (2) T and U starts with T_A and U_A, respectively, upon the packet's arrival to the switch.

With the above discussion, it is clear that we can use a slightly variant of the original MUCFA to perform scheduling in a CIOQ switch and yields identical output behavior as an OQ switch with FIFO scheduling policy. The following is the slightly extended version of MUCFA, which we call *smallest departure time first algorithm* (SDTFA) so as to distinguish it from MUCFA.

Algorithm 2 (SDTFA)

1. At the beginning of each phase, each output tries to obtain its packet with the smallest departure time from the input.

2. If more than one output request the same input, then the input will grant to that output whose packet has the smallest departure time. If there is a tie between two or more outputs, then the output with the smallest port number wins.

3. Outputs that lose such contention at an input will try to obtain their packet with the next smallest departure time from another (unmatched) input port.

4. When no more matching of inputs and outputs is possible, packets are transferred and SDTFA goes to the next phases (Step 1). □

Note that once the departure time D is assigned to the packet upon arrival, it will not change over time and there is no need to update this value (as in the case for urgency under the MUCFA) at the end of each time slot. We will show that such extension not only simplifies the original MUCFA in implementation for mimicking OQ switch under FIFO scheduling, but it also sets the stage for our design of scheduling algorithm using VTRS to achieve the same performance as WFQ for end-to-end delay guarantee (which will soon be made clear).

Theorem 3 *An $N \times N$ CIOQ switch operating under SDTFA and speedup $S \geq 4$, can behave identically to a FIFO OQ switch, regardless of input traffic patterns and for arbitrary switch size N.* □

The proof of Theorem 3 is almost identical to that for Theorem 1, which is given in [11].

Similarly, the SDTFA can be extended to the so-called *monotone* scheduling policy as in the case of MUCFA. The following definition of *expected departure time* (EDT) extends the definition for departure time (D) and sets the stage to extend SDTFA for monotone output scheduling policies.

Definition 4 *The expected departure time, $EDT_p(t)$, of a packet p at an time slot t is the time it would depart from the switch if no new packets arrive to the switch after time t.* □

When the output scheduling policy is FIFO, the $EDT_p(t)$ of packet p is the same as the its departure time D and does not change at the end of each time slot. This need not be the case in a general monotone output scheduling policy since new packets may be pushed into the output queue and take precedence over an existing packet, causing its EDT to increase.

Let SDTFA-E be the algorithm that during any phase of time slot t schedules the transfer of packets from inputs to outputs in exactly the same manner as SDTFA, except for basing its scheduling decisions on a packet's expected departure time EDT instead of its departure time U. Then we have the following theorem, which extends Theorem 3.

Theorem 4 *An $N \times N$ CIOQ switch operating under SDTFA-E and a speedup $S \geq 4$ can behave identically to an OQ switch employing a monotone output scheduling policy, regardless of input traffic patterns and for arbitrary switch size N.* □

The proof of Theorem 4 is similar to the proof for MUCFA-E (see [11]) and is omitted here due to paper length limitation.

4.2 Mimicking an OQ Switch with $C_{\mathcal{S}}$VC Scheduling Policy

Lemma 1 *Under VTRS, the core stateless virtual clock ($C_{\mathcal{S}}$VC) scheduling policy is monotone.* □

Proof. Recall that under VTRS, $C_{\mathcal{S}}$VC scheduler services packets in the order of their *virtual finish time.* For any packet $p^{j,k}$ traversing the OQ switch $\mathcal{S}$, let $\tilde{\omega}^{j,k}$

be the virtual time carried by $p^{j,k}$ as it enters $\mathcal{S}$, and $\tilde{d}^{j,k} = \frac{L^{j,k}}{r^j} + \delta^{j,k}$ be its virtual delay. Then the virtual finish time $\tilde{\nu}^{j,k}$ of $p^{j,k}$ is given by $\tilde{\omega}^{j,k} + \tilde{d}^{j,k}$. Therefore, at the output port of an OQ switch, an arriving packet is pushed into the appropriate location based on its virtual finish time $\tilde{\nu}^{j,k}$ while packets depart only from front. Note that newly arriving packets cannot change the position of the existing packet relative to another. By Definition 2 for monotone scheduling policy, the lemmas is proved. □

The following is the scheduling algorithm used by a CIOQ switch to mimic an OQ switch with $C_{\mathcal{G}}$VC scheduling. As expected, this algorithm is almost identical to the SDTFA-E, except that it schedules packets based on their virtual finish time $\tilde{\nu}^{j,k}$ instead of its expected departure time EDT. We call this algorithm *smallest virtual finish time first algorithm* (SVFTFA) so as to distinguish it from SDTFA-E.

Algorithm 3 (SVFTFA)

1. At the beginning of each phase, each output tries to obtain its packet with the smallest virtual finish time from the input.

2. If more than one output request the same input, then the input will grant to that output whose packet has the smallest virtual finish time. If there is a tie between two or more outputs, then the output with the smallest port number wins.

3. Outputs that lose such contention at an input will try to obtain their packet with the next smallest virtual finish time from another (unmatched) input port.

4. When no more matching of inputs and outputs is possible, packets are transferred and SVFTFA goes to the next phases (Step 1). □

There is a difference between SVFTFA and the SDTFA-E. Under the SDTFA-E, the expected departure time $EDT(t)$ is used by the monotone scheduler for the shadow OQ switch and packet will depart *precisely* at $EDT(t)$ if no new packets arrive at time t. But $C_{\mathcal{G}}$VC uses virtual finish time $\tilde{\nu}^{j,k}$ for scheduling and the actual packet departure time $\hat{f}^{j,k}$, in general, is not the same as the packet's virtual finish time $\tilde{\nu}^{j,k}$, but rather, $\hat{f}^{j,k} \leq \tilde{\nu}^{j,k} + \frac{L^{*,max}}{C}$. However, this does not cause any problem for SVFTFA in a CIOQ switch to mimic an OQ switch with $C_{\mathcal{G}}$VC scheduler *since both $C_{\mathcal{G}}$VC and SVFTFA use the same virtual finish time in scheduling and thus the relative ordering of packets is the same under both schedulers.*

Theorem 5 *An $N \times N$ CIOQ switch operating under SVFTFA and speedup $S \geq 4$, can behave identically to a $C_{\mathcal{G}}$VC OQ switch, regardless of input traffic patterns and for arbitrary switch size N.* □

The proof of Theorem 5 is almost identical to that for Theorem 3 and is omitted here due to paper length constraint.

4.3 Equivalence of $C_{\mathcal{G}}$VC and WFQ for OQ Switch in Providing End-to-end Delay Guarantee

Recall that the VTRS provides a unifying framework to formalize the per-hop behavior of a core switch or router and to quantify its ability to provide delay guarantees. Under VTRS, the scheduling mechanism of the core router can be abstracted into a scheduling blackbox characterized by an error term. Such simple abstraction enabled us to derive end-to-end delay bounds for flows traversing an arbitrary concatenation of such scheduling blackboxes. Furthermore, the VTRS is a unifying scheduling framework that can accommodate *both* core stateless and stateful scheduling algorithms.

1. The VTRS can be used to design new core stateless scheduling algorithm where the VTRS (more precisely, the virtual finish time) is directly used in packet scheduling, e.g., $C_{\mathcal{G}}$VC.

2. In the case when the VTRS is not used in packet scheduling in stateful scheduling algorithms, e.g., WFQ, the switch performs a per-hop virtual time update mechanism to maintain the continual progression of the virtual time embodied in the packet virtual time stamps. In order words, the node updates the virtual time in the packet header so that the property of the virtual time is preserved for future hops and thus provide end-to-end delay guarantee.

The following lemma shows that in terms of supporting end-to-end delay guarantee, a $C_{\mathcal{G}}$VC OQ switch has the same performance as a WFQ OQ switch because both of them have the same error term.

Lemma 2 *In terms of providing end-to-end delay guarantee, under the VTRS, an OQ switch employing the $C_{\mathcal{G}}$VC scheduling algorithm has the same performance as an OQ switch employing the WFQ scheduling algorithm.* □

Proof. Since under the VTRS, both the $C_{\mathcal{G}}$VC and the WFQ have the same error term $\Psi = L^{*,max}/C$, an OQ switch employing either $C_{\mathcal{G}}$VC or WFQ will contributes the same worst case per-hop delay. Therefore, in terms of overall end-to-end delay performance, an OQ switch with $C_{\mathcal{G}}$VC scheduling algorithm has the same performance as an OQ switch under WFQ. □

Note that although the worst case delay performance of an OQ switch is the same under either the $C_{\mathcal{G}}$VC scheduler or WFQ scheduler, the output processes of the switch, in general, are not identical.

4.4 Main Result

With the above results in this section, we are now ready to present the following main result.

Theorem 6 *In terms of providing end-to-end delay guarantee, an $N \times N$ CIOQ switch operating under SVFTFA using VTRS and speedup $S \geq 4$ has the same performance as an OQ switch employing the WFQ scheduling algorithm.* □

Proof. The theorem is proved by combining Theorem 5 and Lemma 2. □

Note that Theorem 6 does not, in general, guarantee a CIOQ switch to *behave identically* to an OQ switch under WFQ scheduling. Although the CIOQ switch under SVFTFA can mimic an OQ switch with $C_{\mathcal{S}}$VC scheduler (i.e., behave identically), an OQ switch with $C_{\mathcal{S}}$VC does not, in general, have identical behavior as an OQ switch with WFQ scheduler. In this regard, one might be a little bit disappointed as our original design objective is to find a practical implementation for a CIOQ switch to behave identically to an OQ switch with WFQ scheduler. But we point out that the ultimate objective for an OQ switch employing WFQ scheduler is to provide end-to-end delay guarantee and in this regard, we have succeeded since the $C_{\mathcal{S}}$VC scheduler under VTRS provides the same end-to-end delay guarantee as the WFQ scheduler.

5 Concluding Remarks

The CIOQ switching architecture has been proposed as a solution to meet the high speed switching and QoS requirements for Internet core nodes. However, many of the existing QoS scheduling algorithms for a CIOQ switch cannot be practically implemented due to problems such as scalability and complexity.

This paper showed how the virtual time reference system (VTRS) can be applied to packet scheduling in a CIOQ switch. The VTRS is a unifying scheduling framework to provide scalable support for guaranteed services. In the context of packet scheduling for a CIOQ switch, we showed that the use of VTRS can eliminate both the scalability and complexity problems associated with existing scheduling algorithms in the literature. More important, we showed that in term of providing end-to-end guaranteed services, packet scheduling for a CIOQ switch using VTRS has the same performance as an output queued (OQ) switch employing weighted fair queueing (WFQ) scheduler.

Acknowledgments

We wish to thank Dr. Toshitaka Tsuda and Dr. Iwao Toda of Fujitsu Laboratories Ltd., Kawasaki, Japan, for their support in our work. Y. T. Hou thanks Prof. Balaji Prabhakar and Shang-Tse Chuang of Stanford University for discussions on matching output queueing with a combined input/output queued switch.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," *RFC 2475,* Internet Engineering Task Force, Dec. 1998.

[2] R. Braden, D. Clark and S. Shenker, "Integrated services in the Internet architecture: an overview," *RFC 1633,* Internet Engineering Task Force, July 1994.

[3] S.-T. Chuang, A. Goel, N. McKeown and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE J. Select. Areas in Commun.,* vol. 17, no. 6, pp. 1030–1039, June 1999.

[4] D. Clark, S. Shenker and L. Zhang, "Supporting real-time applications in an integrated services packet network: architecture and mechanisms," *Proc. ACM SIGCOMM,* August 1992, Baltimore, MD.

[5] A. Demers, S. Keshav and S. Shenker, "Analysis and simulations of a fair queueing algorithm," in *Proc. ACM SIGCOMM,* Austin, TX, Sept. 1989, pp. 1–12.

[6] N. Figueira and J. Pasquale, "An upper bound on delay for the virtual clock service discipline," *IEEE/ACM Trans. on Networking,* vol. 3, no. 4, pp. 399–408, August 1995.

[7] M. Karol, M. G. Hluchyj and S. P. Morgan, "Input vs. output queueing on a space-division packet switch," *IEEE Trans. on Commun.,* vol. 35, no. 12, pp. 1347–1356, Dec. 1987.

[8] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. on Networking,* vol. 7, no. 2, pp. 188–201, April 1999.

[9] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Trans. on Networking,* vol. 1, no. 3, pp. 344–357, June 1993.

[10] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," *IEEE/ACM Trans. on Networking,* vol. 2, no. 2, pp. 137–150, April 1994.

[11] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output queued switching," *Automatica,* vol. 35, issue 12, pp. 1909–1920, Dec. 1999.

[12] S. Shenker, C. Partridge and R. Guerin, "Specification of guaranteed quality of service," *RFC 2212,* Internet Engineering Task Force, Sept. 1997.

[13] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. on Networking,* vol. 6, no. 5, pp. 611–624, Oct. 1998.

[14] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," *Proc. ACM SIGCOMM,* Sept. 1999, Cambridge, MA

[15] Z.-L. Zhang, Z. Duan and Y. T. Hou, "Virtual time reference system: a unifying scheduling framework for scalable support of guaranteed services," submitted to *IEEE J. on Select. Areas in Commun.* (special issue on Internet QoS).