On Guaranteed Smooth Switching for Buffered Crossbar Switches

Si-Min He, *Member, IEEE*, Shu-Tao Sun, Hong-Tao Guan, Qiang Zheng, You-Jian Zhao, and Wen Gao, *Senior Member, IEEE*

***Abstract*—Scalability considerations drive the evolution of switch design from output queueing to input queueing and further to combined input and crosspoint queueing (CICQ). However, CICQ switches with credit-based flow control face new challenges of scalability and predictability. In this paper, we propose a novel approach of rate-based smoothed switching, and design a CICQ switch called the smoothed buffered crossbar or sBUX. First, the concept of smoothness is developed from two complementary perspectives of covering and spacing, which, commonly known as fairness and jitter, are unified in the same model. Second, a smoothed multiplexer sMUX is designed that allocates bandwidth among competing flows sharing a link and guarantees almost ideal smoothness for each flow. Third, the buffered crossbar sBUX is designed that uses the scheduler sMUX at each input and output, and a two-cell buffer at each crosspoint. It is proved that sBUX guarantees 100% throughput for real-time services and almost ideal smoothness for each flow. Fourth, an on-line bandwidth regulator is designed that periodically estimates bandwidth demand and generates admissible allocations, which enables sBUX to support best-effort services. Simulation shows almost 100% throughput and multi-microsecond average delay. In particular, neither credit-based flow control nor speedup is used, and arbitrary fabric-internal latency is allowed between line cards and the switch core, simplifying the switch implementation.**

***Index Terms*—Buffered crossbar, scheduling, smoothness, switches.**

I. Introduction

THE EVOLUTION of switch design is driven by two challenging and often conflicting goals: scalability, and predictability. While output-queued (OQ) switches can provide perfect predictability [44], they are not scalable since an $N \times N$ OQ switch requires that memory runs at least N times faster than external links; i.e., a *speedup* of at least N is needed. In contrast, input-queued (IQ) crossbar switches require no such a large memory speedup, and their predictability has been greatly improved by extensive research in the past decade [2], [3] [5], [7], [10], [12], [22], [24], [25], [27], [28], [39]. But IQ crossbar switches require a scheduler of high computing complexity, either to provide 100% throughput for best-effort or real-time services, or to provide perfect or performance emulation of OQ switches. In particular, a speedup of two seems inevitable both in principle and in practice, since input and output contentions have to be resolved *simultaneously*. To further improve scalability, the combined input- and crosspoint-queued (CICQ) switch is proposed, and is the focus of this paper.

The CICQ switch is usually implemented as a buffered crossbar, with virtual output queues (VOQs) at inputs and limited buffers at crosspoints. By distributed input and output scheduling at entries and exits to the switch core, a CICQ switch resolves input and output contentions *separately* by $2N$ simple schedulers, in a distributed and parallel manner. This implies better scalability than an IQ crossbar switch. Since the on-chip crosspoint buffer is small, flow control is needed to prevent buffer overflow and cell loss. At present the credit-based flow control is predominantly used in CICQ switches to coordinate input and output scheduling [23].

CICQ switches attract intensive research that is conducted in the same way of crossbar study: throughput-oriented, OQ switch emulation-oriented, and guarantee-oriented.

In throughput-oriented research, various combinations of simple input and output schedulers have been studied by simulation, showing 100% throughput under uniform workloads, but less than 100% under non-uniform workloads [19], [29] [33], [37], [38], [43]. It is proved that a CICQ switch with longest-queue-first input schedulers, round-robin output schedulers, and one-cell crosspoint buffers, achieves 100% throughput under any admissible workload with each input-output pair loaded no more than $1/N$ [19]. *With a speedup of two*, it is proved that a buffered crossbar employing any work-conserving input and output schedulers can achieve 100% throughput [11], even under asynchronous packet-based scheduling [34]; by adopting a finite crosspoint buffer with size $B > N$, the speedup can be reduced to $\lceil 2B/(2B-N) \rceil$ [14].

In OQ switch emulation-oriented research, by extension of the techniques used for crossbar switches [10], [22], it is proved that buffered crossbar switches, *with a speedup of two*, can emulate an OQ switch that employs any push-in, first-out queueing policy such as first-in-first-out and strict-priority, or just keep the switch work-conserving [11], [26], [42].

In guarantee-oriented research, a distributed packet fair queueing buffered crossbar architecture is proposed, and *with modest speedup support*, its bandwidth and delay performances are shown by simulation to approximate those of an OQ switch employing fair queueing [40]. The technique of network calculus is applied to determining the amount of bandwidth that must be allocated to a flow so as to guarantee its delay upper bound [13].

Manuscript received April 9, 2006; revised November 4, 2006 and January 26, 2007; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Ajmone Marsan. This work was supported in part by the Natural Science Foundation of China under Grants 69983008 and 90604029, and in part by the Institute of Computing Technology, Chinese Academy of Sciences, under Grant 20056090. A preliminary version of this paper appeared in the ACM SIGMETRICS 2005, Banff, Canada.

S.-M. He and Q. Zheng are with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China (e-mail: smhe@ict.ac.cn; qzheng@ict.ac.cn).

S.-T. Sun is with the School of Computer and Software, Communication University of China, Beijing 100024, China (e-mail: stsun@cuc.edu.cn).

H.-T. Guan and Y.-J. Zhao are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100086, China (e-mail: ght@csnet1.cs.tsinghua.edu.cn; zhaoyj@csnet1.cs.tsinghua.edu.cn).

W. Gao is with the Institute of Digital Media, Peking University, Beijing 100871, China (e-mail: wgao@pku.edu.cn).

Digital Object Identifier 10.1109/TNET.2007.900402

In particular, to guarantee this allocated bandwidth for this flow, the number of credits required, or the crosspoint buffer capacity, shall be at least the product of the required bandwidth and the total fabric-internal round-trip time of this flow.

Actually, the coupled effect of fabric-internal latency and credit-based flow control is often neglected by most previous research. Since a multi-terabit switch must be distributed over multiple racks, the fabric-internal latency is increased considerably along with the distance between line cards and the switch core [8], [31]. For example, a 4-Tb/s CICQ packet switch is distinguished with a large fabric-internal latency by packaging up to 256 10-Gb/s (OC-192) line cards into multiple racks, with a distance of tens to hundreds of feet between line cards and the switch core [1]. Large fabric-internal latency, if not properly dealt with, degrades sharply the performance of a crossbar switch [32]. With credit-based flow control, an arbitrarily small crosspoint buffer can guarantee no cell loss; but to guarantee bandwidths to flows [13], or to guarantee input work conservation and output full utilization [1], [20], the size of each crosspoint buffer shall be at least the product of the link speed and the fabric-internal round-trip time, and the size of total crosspoint memory will be N^2 times large since there are N^2 crosspoint buffers in a switch.

Therefore, CICQ switches with credit-based flow control face new challenges of scalability and predictability. Large crosspoint buffer, induced by large fabric-internal latency and fast link speed, is surely a bottleneck to scalability. *Without speedup*, it is still open whether CICQ switches can guarantee 100% throughput for either best-effort or real-time services.

In this paper, we propose a design approach of *rate-based smoothed switching* to tackle these problems. Rate information is more compact and hence more scalable than credit information. It can feasibly be obtained by admission control for real-time services, or by on-line bandwidth demand estimation and allocation for best-effort services. With additional rate information, predictability of switch design can be enhanced. In particular, for CICQ switches, if rate-based *smoothed* schedulers are placed at inputs and outputs, filling and draining crosspoint buffers *smoothly*, then a small buffer will suffice, regardless of fabric-internal latency or link speed. This will break the bottleneck of large crosspoint buffers, eliminate the credit-based feedback flow control, and reinforce predictability.

Specifically, Section II develops the concept of smoothness from two intuitive and complementary viewpoints: covering smoothness and spacing smoothness, which are then unified in the same model. Section III proposes a smoothed multiplexer sMUX to solve the smooth multiplexing problem, with almost ideal smoothness bounds obtained for each of the flows sharing a link. Section IV proposes a smoothed buffered crossbar sBUX, which uses scheduler sMUX at inputs and outputs, and a buffer of just *two* cells at each crosspoint. It is proved that sBUX can use 100% of the switch capacity to support real-time services, with deterministic guarantees of bandwidth and fairness, delay and jitter bounds for each flow. To support best-effort services, Section V introduces a bandwidth regulator to periodically deliver an admissible bandwidth matrix to sBUX by on-line bandwidth demand estimation and allocation. The two-cell crosspoint buffers are still sufficient to guarantee no cell loss even during dynamic bandwidth updating. Section VI conducts simulation to show that sBUX coupled with the bandwidth regulator can deliver a throughput of almost 100% and an average delay of multiple microseconds. In particular, neither credit-based flow control nor speedup is needed, and arbitrary fabric-internal latency is allowed between line cards and the switch core. Section VII concludes the paper with discussion of the rate-based control scheme and the smooth switching problems for some leading switches. All proofs of theorems are placed in the Appendix.

II. Smoothness

This section defines the concept of smoothness.

There are n flows of fixed-size cells sharing a link of bandwidth r; each flow f_i is allocated a bandwidth r_i, where $r_i \geq 0$ and $\Sigma_i r_i \leq r$. This specifies an instance $(r; r_1, r_2, \ldots, r_n)$, which can be reduced to its normal form $(1; w_1, w_2, \ldots, w_n)$, abbr. $(w_1, w_2, \ldots, w_n)$, in which $w_i = r_i/r \geq 0$ and $\Sigma w_i \leq 1$.

Time is slotted, with slot t denoting the real interval $[t, t+1)$, and slot interval$[t_1, t_2)$ denoting the slot set $\{t_1, t_1+1, \ldots, t_2-1\}$. A schedule or scheduler S for an instance $(w_1, w_2, \ldots, w_n)$ is a function $S : [t_1, t_2) \rightarrow \{\perp, \tau_1, \tau_2, \ldots, \tau_n\}$, mapping slots to symbols or cell services; symbol $\perp$ stands for an idle cell service and symbol τ_i stands for a cell service to flow f_i.

The *smooth multiplexing problem* (SMP) is to generate a smooth schedule such that cell services to each flow f_i are distributed evenly in the whole sequence. Intuitively, in an ideally smooth schedule for an SMP instance $(w_1, w_2, \ldots, w_n)$, any interval of l consecutive slots should cover $(l \cdot w_i)$ number, or in practice, either $\lfloor l \cdot w_i \rfloor$ or $\lceil l \cdot w_i \rceil$ number of cell services to flow f_i. In a complementary view, successive $(s+1)$ number of cell services to flow f_i should be spacing (s/w_i) slots apart, or either $\lfloor s/w_i \rfloor$ or $\lceil s/w_i \rceil$ slots apart. Such different and intuitive views of covering and spacing, along with the integral constraint, shall be taken into account during formalization.

A. Covering

Let $Cover(S, \tau_i, t, l)$, abbr. $Cover_i(t, l)$, denote the number of cell services to flow f_i that are scheduled by scheduler S inside slot interval $[t, t+l)$. We investigate the whole spectrum of $Cover_i(t, l)$ and its worst-case deviation from an ideal distribution over all possible slot intervals $[t, t+l)$.

Given a slot interval $L = [t_1, t_2)$, $t_1 < t_2$, which could be finite $(t_2 < +\infty)$ or infinite $(t_2 = +\infty)$, the minimum and the maximum covers at length l within this interval can be defined as follows:

$$\begin{aligned}&Minimum\ cover\ cvr\,(S, \tau_i, l; [t_1, t_2))\\&\quad = min_t\{Cover_i(t, l) | [t, t+l) \subseteq [t_1, t_2)\}, \text{ abbr. } cvr_i(l);\\&Maximum\ cover\ CVR\,(S, \tau_i, l; [t_1, t_2))\\&\quad = max_t\{Cover_i(t, l) | [t, t+l) \subseteq [t_1, t_2)\}, \text{ abbr. } CVR_i(l).\end{aligned}$$

We measure covering smoothness of the actual distribution of cell services to flow f_i within $[t_1, t_2)$ by the following two worst-case covering deviations from the ideal:

$$\begin{aligned}cvr\text{-}dev_i &= max_l\{\lfloor l \cdot w_i \rfloor - cvr_i(l) | 0 \leq l \leq t_2 - t_1\};\\CVR\text{-}dev_i &= max_l\{CVR_i(l) - \lceil l \cdot w_i \rceil | 0 \leq l \leq t_2 - t_1\}.\end{aligned}$$

By definition, $cvr\text{-}dev_i \geq 0$, $CVR\text{-}dev_i \geq 0$. See Table I for illustration. Within interval $[t_1, t_2)$, if $cvr\text{-}dev_i = CVR\text{-}dev_i = 0$, or equivalently, if $\lfloor l \cdot w_i \rfloor \leq Cover_i(t, l) \leq \lceil l \cdot w_i \rceil$ for any

TABLE I
COVERING AND SPACING PROPERTIES OF τ_3 IN $\mathrm{S}[0,6) = (\tau_1, \tau_2, \tau_3, \tau_1, \tau_3, \tau_3)$, $w_3 = 4/6$

Defined in [0, 6)	$cvr_3(l)$	*ideal* $=\lfloor l \cdot w_3 \rfloor$	$cvr\text{-}dev_3(l)$ $= ideal - cvr_3(l)$	$CVR_3(l)$	*ideal* $=\lceil l \cdot w_3 \rceil$	$CVR\text{-}dev_3(l)$ $= CVR_3(l) - ideal$
$l = 0$	0	0	0	0	0	0
$l = 1$	0	0	0	1	1	0
$l = 2$	0	1	1	2	2	0
$l = 3$	1	2	1	2	2	0
$l = 4$	1	2	1	3	3	0
$l = 5$	2	3	1	3	4	–1
$l = 6$	3	4	1	3	4	–1
			$cvr\text{-}dev_3 = 1$			$CVR\text{-}dev_3 = 0$
Defined in [2, 6)	$spc_3(s)$	*ideal* $=\lfloor s / w_3 \rfloor$	$spc\text{-}dev_3(s)$ $= ideal - spc_3(s)$	$SPC_3(s)$	*ideal* $=\lceil s / w_3 \rceil$	$SPC\text{-}dev_3(s)$ $= SPC_3(s) - ideal$
$s = 0$	0	0	0	0	0	0
$s = 1$	1	1	0	2	2	0
$s = 2$	3	3	0	3	3	0
			$spc\text{-}dev_3 = 0$			$SPC\text{-}dev_3 = 0$

interval $[t, t+l) \subseteq [t_1, t_2)$, then the distribution of cell services to flow f_i is ideal from the covering point of view.

B. Spacing

Let $Pos(S, \tau_i, j)$, abbr. $Pos_i(j)$, denote the slot of the jth cell service to flow f_i scheduled by scheduler S, and let $Space(S, \tau_i, j, s)$, abbr. $Space_i(j, s)$, denote an s-step space, or the number of slots between the positions of the jth and $(j+s)$th cell services to flow $f_i : Space_i(j, s) = Pos_i(j+s) - Pos_i(j)$. Then given an interval $[Pos_i(j_1), Pos_i(j_2) + 1)$, $j_1 \leq j_2$, either finite $(j_2 < +\infty)$ or infinite $(j_2 = +\infty)$, the minimum and the maximum spaces at step s within this interval can be defined as follows:

$$\begin{aligned} &Minimum\ space\ spc\,(S, \tau_i, s; [j_1, j_2]) \\ &\quad = min_j\{Space_i(j, s) | j_1 \leq j, j+s \leq j_2\}, \text{ abbr. } spc_i(s); \\ &Maximum\ space\ SPC\,(S, \tau_i, s; [j_1, j_2]) \\ &\quad = max_j\{Space_i(j, s) | j_1 \leq j, j+s \leq j_2\}, \text{ abbr. } SPC_i(s). \end{aligned}$$

We measure spacing smoothness of the actual distribution of cell services to flow f_i within $[Pos_i(j_1), Pos_i(j_2) + 1)$ by the following two worst-case spacing deviations from the ideal:

$$\begin{aligned} spc\text{-}dev_i &= max_s\{\lfloor s/w_i \rfloor - spc_i(s) | 0 \leq s \leq j_2 - j_1\}; \\ SPC\text{-}dev_i &= max_s\{SPC_i(s) - \lceil s/w_i \rceil | 0 \leq s \leq j_2 - j_1\}. \end{aligned}$$

By definition, $spc\text{-}dev_i \geq 0$, $SPC\text{-}dev_i \geq 0$. See Table I for illustration. Within interval $[Pos_i(j_1), Pos_i(j_2) + 1)$, if $spc\text{-}dev_i = SPC\text{-}dev_i = 0$, or equivalently, if $\lfloor s/w_i \rfloor \leq Space_i(j, s) \leq \lceil s/w_i \rceil$ for all j and s subject to $j_1 \leq j$ and $j + s \leq j_2$, then the distribution of cell services to flow f_i is ideal from the spacing point of view.

C. Bridge Between Covering and Spacing

Before we proceed, we should first notice the different domains of definition for covering and spacing. Suppose covering properties are defined in interval $[t_1, t_2)$, which contains $(s+1)$ number of cell services to flow f_i located at $Pos_i(j_1)$ through $Pos_i(j_1+s)$, then spacing properties can be defined only within interval $[Pos_i(j_1), Pos_i(j_1+s)+1)$, which is normally a proper subset of $[t_1, t_2)$. Intuitively the properties defined in the superset should be able to determine the properties defined in the subset, but not vice versa.

Theorem 2.1: *In a schedule for an SMP instance, for any flow f_i, and any nonnegative integers s, l, d, and m,*

$$\begin{aligned} &(1)\ cvr_i(l) \geq m \Rightarrow SPC_i(m) \leq l \\ &(2)\ SPC_i(m) \leq l \Rightarrow cvr_i(l) \geq m \\ &(3)\ CVR_i(d) \leq s \Rightarrow spc_i(s) \geq d \\ &(4)\ spc_i(s) \geq d \Rightarrow CVR_i(d) \leq s \end{aligned}$$

where the right-hand side of $\Rightarrow$ is defined within the interval in which the left-hand side of $\Rightarrow$ is defined.

Theorem 2.2: *In a schedule for an SMP instance, for any flow f_i,*

$$\begin{aligned} &(1)\ SPC\text{-}dev_i \leq \lceil cvr\text{-}dev_i / w_i \rceil \\ &(2)\ spc\text{-}dev_i \leq \lceil CVR\text{-}dev_i / w_i \rceil \\ &(3)\ cvr\text{-}dev_i \leq \lceil SPC\text{-}dev_i \cdot w_i \rceil \\ &(4)\ CVR\text{-}dev_i \leq \lceil spc\text{-}dev_i \cdot w_i \rceil \end{aligned}$$

where the left-hand side of $\leq$ is defined within the interval in which the right-hand side of $\leq$ is defined.

Theorem 2.2 describes the relationship between covering smoothness and spacing smoothness, revealing their consistency and correspondence. It has a corollary as follows.

Corollary 2.3: *In a schedule for an SMP instance, for any flow f_i,*

$$\begin{aligned} &(1)\ cvr\text{-}dev_i = CVR\text{-}dev_i = 0 \\ &\qquad \Rightarrow SPC\text{-}dev_i = spc\text{-}dev_i = 0 \\ &(2)\ SPC\text{-}dev_i = spc\text{-}dev_i = 0 \\ &\qquad \Rightarrow cvr\text{-}dev_i = CVR\text{-}dev_i = 0, \end{aligned}$$

where the right-hand side of $\Rightarrow$ is defined within the interval in which the left-hand side of $\Rightarrow$ is defined.

Corollary 2.3 reveals the correspondence between ideal covering and ideal spacing, strengthening the rationality of smoothness definitions. Given a schedule S for an SMP instance, if $cvr\text{-}dev_i = CVR\text{-}dev_i = 0$, then the schedule is called *ideal for flow f_i*; if the schedule is *ideal for every flow*, then the schedule is called *ideal*. In general, an ideal schedule might not exist, e.g., for the SMP instance (2/6, 3/6, 1/6). But it is easy to generate an ideal schedule only for a single flow, or for two flows whose normalized bandwidths add up to 1, as shown next.

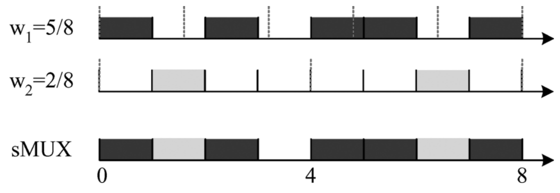


Fig. 1. The sMUX schedule for SMP instance (5/8, 2/8) with common initiation time 0. The dashed lines are eligible times and deadlines for each cell service.

Theorem 2.4: *SMP instance (w_1, w_2) with $w_1 + w_2 = 1$ always has an ideal schedule, which can be constructed by allocating cell services to flow f_1 to slots $\lceil x + j/w_1 \rceil$ for all integer j, and cell services to flow f_2 to the slots left, where x is an arbitrary but fixed real number.*

While the concept of smoothness was defined *within a periodic frame* as certain covering discrepancy in [36], their definition did not consider the viewpoint of *spacing*, let alone the *relationship* between covering and spacing, in a *general* sequence. No formal definitions of smoothness are given in [21]. To our best knowledge, our definitions of smoothness are most general and comprehensive.

III. Smoothed Multiplexer sMUX

This section designs and analyses an algorithm called *smoothed multiplexer* or sMUX.

A. Algorithm sMUX

Assume an SMP instance $(w_1, w_2, \dots, w_n)$ is given. Besides the normalized bandwidth w_i, each flow f_i is additionally associated with an initiation time I_i to mark the earliest time slot the scheduler is ready to schedule flow f_i. This is to reflect the dynamic scheduling of newly admitted flows in practical applications. Starting from time I_i, the jth $(j = 1, 2, 3 \dots)$ cell service to flow f_i is *eligible* at time $e_{i,j} = I_i + (j-1)/w_i$, and is expected to finish before *deadline* $d_{i,j} = I_i + j/w_i$. That is, the jth cell service for flow f_i is expected to be allocated in the jth window $[I_i + (j-1)/w_i, I_i + j/w_i)$.

If resource allocation can be in units of arbitrarily small size rather than in units of slot, then an *earliest deadline first* or EDF scheduler shall *feasibly* schedule the set of flows (i.e., all the requirements of eligible time, service time and deadline shall be met) since $\Sigma_i w_i \leq 1$ [41]. But solutions to SMP require that each flow be serviced in units of an integral time slot, which makes it impossible to meet all the requirements stated before. Algorithm sMUX, as an EDF scheduler adapted to the integral constraint, tries to achieve the best approximation.

Algorithm sMUX: *At each time slot t, among those flows that are eligible for scheduling, i.e., their next cell services have eligible times no later than t ($e_{i,j} \leq t$, or equivalently, $\lceil e_{i,j} \rceil \leq t$), allocate slot t to the flow with the earliest upper-rounded deadline $\lceil d_{i,j} \rceil$; ties are broken arbitrarily. When no flow is eligible, slot t is left idle.*

Fig. 1 is an illustration of how sMUX works on an SMP instance (5/8, 2/8) with all $I_i = 0$.

It is obvious that sMUX provides the jth cell service for flow f_i after its eligible time $e_{i,j}$. The following theorem shows that the service finishes before $\lceil d_{i,j} \rceil$.

Theorem 3.1: *In a sMUX schedule for an SMP instance $(w_1, w_2, \dots, w_n)$ with initiation times $(I_1, I_2, \dots, I_n)$, the jth ($j = 1, 2 \dots$) cell service to flow f_i is allocated within the slot interval $[\lceil e_{i,j} \rceil, \lceil d_{i,j} \rceil)$, where $e_{i,j} = I_i + (j-1)/w_i$, and $d_{i,j} = I_i + j/w_i$.*

B. Guaranteed Smoothness of sMUX

Theorem 3.2 below characterizes the guaranteed smoothness bounds of sMUX. Owing to introduction of the individual initiation time I_i, the scheduler is ready to schedule each flow at possibly different times. Hence, the covering properties concerning flow f_i should be calculated from slot I_i onward.

Theorem 3.2: *Given a sMUX schedule for an SMP instance $(w_1, w_2, \dots, w_n)$ with initiation times $(I_1, I_2, \dots, I_n)$. Then for any flow f_i, interval length l, and step size s, the sMUX schedule has the following properties:*

(1) $\lfloor l \cdot w_i \rfloor \leq Cover(S, \tau_i, I_i, l) \leq \lfloor (l-1) \cdot w_i \rfloor + 1$

(2) $\lfloor (l+1) \cdot w_i \rfloor - 1 \leq cvr_i(l) \leq CVR_i(l) \leq \lceil (l-1) \cdot w_i \rceil + 1$

(3) $cvr\text{-}dev_i \leq 1,\ CVR\text{-}dev_i \leq 1$

(4) $\lfloor (s-1)/w_i \rfloor + 1 \leq spc_i(s) \leq SPC_i(s) \leq \lceil (s+1)/w_i \rceil - 1$

(5) $spc\text{-}dev_i \leq \lfloor 1/w_i \rfloor,\ SPC\text{-}dev_i \leq \lfloor 1/w_i \rfloor$.

The covering deviation of sMUX is at most one cell service, which could be outperformed only by an ideal schedule if it exists. In this sense the sMUX schedule is almost ideal or optimal. When an ideal schedule does not exist, which is the normal case, the sMUX schedule is optimal. The constant deviation also indicates that compared with the spacing deviation, the covering deviation seems to be a normalized measure of smoothness.

By eligible time control, cells of flow f_i are admitted into sMUX at times I_i, $I_i + \lceil 1/w_i \rceil, I_i + \lceil 2/w_i \rceil, \dots$, with all spacing and covering deviations equal to zero. If sMUX only introduces a constant delay to each cell, then spacing deviations of leaving cell sequences should remain zero. Therefore, any non-zero spacing deviations are actually the delay jitters, resulting from variable delays introduced to cells by sMUX.

Simply speaking, sMUX combines the guarantee of circuit multiplexing and the flexibility of packet multiplexing [18]. The frame-based circuit multiplexing guarantees dedicated bandwidth and fixed delay (comparable with the frame duration), but suffers from a service granularity problem: a small frame might waste bandwidth while a large frame complicates frame scheduling and storage. In contrast, packet multiplexing allows more flexible bandwidth sharing, but the worst-case delay might be unpredictable. For real-time services, e.g., continuous media, we need flexible bandwidth allocation and predictable worst-case delay and jitter bounds. The scheduler sMUX allows arbitrary and almost exact sharing of the link bandwidth; the implicit huge frame reference can be considered to be compactly embedded in the slot-by-slot sMUX algorithm. The worst-case delay and jitter, however, are independent of the length of the implicit frame reference.

In terms of fast hardware implementation, the minimum of 256 24-bit numbers can be found in 4.5 ns in 0.18 μm CMOS technology [16]. For comparison, the transmission time of a 64-byte cell in a 40-Gb/s (OC-768) link is 12.8 ns. This strongly suggests that sMUX can feasibly be used in switches with high link speed (~40 Gb/s) and high port density (~256).

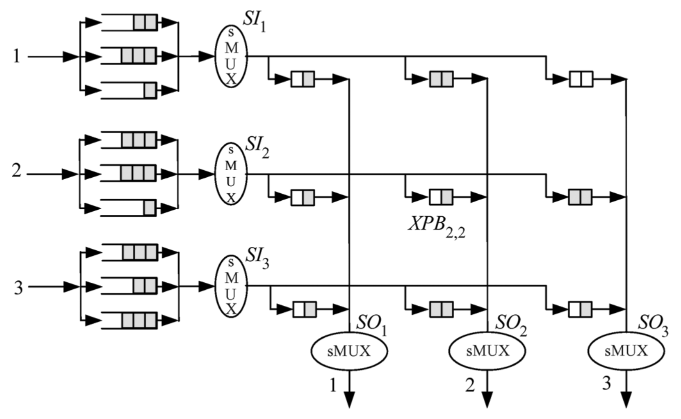


Fig. 2. Architecture of a 3 × 3 buffered crossbar switch sBUX.

IV. Smoothed Buffered Crossbar sBUX

This section describes a CICQ switch called *smoothed buffered crossbar* or $sBUX$, which uses sMUX as both input and output schedulers. It will be shown that a two-cell crosspoint buffer (XPB) suffices to guarantee no cell loss even without credit-based flow control.

A. Switch Model

Consider an $N \times N$ CICQ switch with unit-speed input and output links. For each input-output pair (i,j), the switch has a virtual output queue $VOQ_{i,j}$ at input, and a first-in-first-out crosspoint buffer $XPB_{i,j}$ inside the crossbar. There is a scheduler SI_i for each input i, and there is a scheduler SO_j for each output j; they are all sMUX schedulers. The fabric-internal latency can be arbitrary between line cards and the switch core. See Fig. 2.

Data crossing the switch are organized into flows of fixed-size cells. In this paper we consider unicast flows. Cells of flow $f_{i,j}$ originally arrive at input i, temporarily queue at $VOQ_{i,j}$ and then at $XPB_{i,j}$, and finally go to output j. Associated with flow $f_{i,j}$ is a bandwidth $r_{i,j}$, and bandwidth matrix $R=(r_{i,j})_{N\times N}$ is feasible or admissible, i.e., $\Sigma_i r_{i,j} \leq 1$ for all j and $\Sigma_j r_{i,j} \leq 1$ for all i. For bandwidth-guaranteed real-time services, such a matrix should be available by admission control. For best-effort services, such a matrix is obtained periodically by an on-line bandwidth regulator, to be described in Section V.

Time is slotted with each time slot being the time to transmit one cell at link speed. At each time slot, each input scheduler SI_i will select one VOQ at input i, say $VOQ_{i,k}$, based on the parameters in row i of matrix R, and transmit the head cell to the corresponding crosspoint buffer $XPB_{i,k}$. At the same time slot, each out scheduler SO_j will select one crosspoint buffer, say $XPB_{h,j}$, based on the parameters in column j of matrix R, and transmit the head cell to the corresponding output j. The $2N$ sMUX schedulers make their decisions independently and in parallel, and then the switch executes their decisions synchronously.

What is unique to switch sBUX is that no credit-based flow control is used for each XPB. Specifically, at each time slot, according to sMUX, each input scheduler will provide a cell service to one VOQ, no matter whether this VOQ is backlogged or the corresponding XPB has any vacancy; each output scheduler will provide a cell service to one XPB, no matter whether this XPB is backlogged. Hence, input schedulers and output schedulers are further decoupled.

While the overhead of credit-based flow control is eliminated, the small crosspoint buffers are in danger of overflow. However, the occupancy of each XPB is always upper-bounded by *two* cells, as shown next.

B. XPB Occupancy Analysis

Let us take $XPB_{i,j}$ as our point of view.

In practice, it is reasonable to place SO_j at the switch core and SI_i at the ith line card, with a latency D_i in between. By available hardware technology, this latency D_i can be measured precisely and compensated accordingly such that when SO_j starts to schedule $f_{i,j}$ out of $XPB_{i,j}$ at initiation time $I_{i,j}$, SI_i has already started to schedule $f_{i,j}$ out of $VOQ_{i,j}$ at an earlier initiation time $(I_{i,j}-D_i)$. This is functionally equivalent to placing SI_i and $VOQ_{i,j}$ at where $XPB_{i,j}$ is located, with no latency in between, and starting to schedule $f_{i,j}$ into $XPB_{i,j}$ at the *delayed* initiation time $(I_{i,j}-D_i)+D_i = I_{i,j}$, or at the same time that SO_j starts to schedule $f_{i,j}$ out of $XPB_{i,j}$. Therefore, from now on, we will assume that both SI_i and SO_j, together with $VOQ_{i,j}$, are placed at where $XPB_{i,j}$ is located, with no latency in between. They start scheduling $f_{i,j}$ into and out of $XPB_{i,j}$ with the same rate $r_{i,j}$, at the same initiation time $I_{i,j}$.

Assume that $XPB_{i,j}$ has an infinite capacity and is empty initially (i.e., empty before $I_{i,j}$). We are interested in the *maximum occupancy* of $XPB_{i,j}$ under the operations of SI_i and SO_j, which will be the *minimum capacity* of $XPB_{i,j}$ to guarantee no cell loss. Because the input scheduling and the output scheduling are completely decoupled in a sBUX switch, a key observation is: the *critical situation* to produce maximum occupancy of $XPB_{i,j}$ is when $VOQ_{i,j}$ is always backlogged.

Since both input and output scheduling are rate-based, regardless of cell availability, input and output operations may be *ineffective* when there is no cell available in VOQ or XPB. Specifically, in time interval $[t,t+l)$, $Cover(SI_i,\tau_{i,j},t,l)$ number of input operations on $XPB_{i,j}$ consist of $eCover(SI_i,\tau_{i,j},t,l)$ number of effective ones and $ieCover(SI_i,\tau_{i,j},t,l)$ number of ineffective ones. Similarly, $Cover(SO_j,\tau_{i,j},t,l)$ number of output operations on $XPB_{i,j}$ consist of $eCover(SO_j,\tau_{i,j},t,l)$ number of effective ones and $ieCover(SO_j,\tau_{i,j},t,l)$ number of ineffective ones. Under the critical situation, all input operations are apparently effective, but what happens with output operations and crosspoint buffers?

Theorem 4.1: *Under the critical situation, at most one output operation is ineffective:*

$$ieCover(SO_j,\tau_{i,j},I_{i,j},l) \leq 1.$$

Theorem 4.2: *Let $XPBO_{i,j}$ denote the maximum occupancy of $XPB_{i,j}$, then $XPBO_{i,j}$ is upper-bounded by two cells:*

$$XPBO_{i,j} \leq 2.$$

Theorem 4.2 assumes a fixed bandwidth. What happens if the bandwidth is updated dynamically?

Theorem 4.3: *Assume that once every T slots, the admissible bandwidth matrix is updated with possibly new parameters. Assume that each bandwidth is represented by $r_{i,j}=a_{i,j}/T$,* *where $a_{i,j}$ is an integer. Then $XPBO_{i,j}$ is still bounded by two cells:*

$$XPBO_{i,j} \leq 2.$$

Note that this two-cell XPB suffices for arbitrary fabric-internal latency and arbitrary link speed. In contrast, the size of each crosspoint buffer is 64 cells in [1] or 2 KB in [20] for their respective products of fabric-internal round-trip time and flow or line rate.

C. Guaranteed Performances

The sBUX switch, including the input schedulers, the output schedulers and the crosspoint buffers, is almost equivalent to an OQ switch operated by sMUX, because each sMUX output scheduler of sBUX conducts at most one ineffective service operation when all VOQs are backlogged at inputs (Theorem 4.1), which is negligible for all practical purposes.

If taking this ineffective operation into account, sBUX provides the following worst-case performance guarantees.

Theorem 4.4: *In a sBUX switch, flow $f_{i,j}$ or the channel from input i to output j is guaranteed a service with the following smoothness bounds:*

$$(1)\ cvr\text{-}dev_{i,j} \leq 2,\ CVR\text{-}dev_{i,j} \leq 1$$
$$(2)\ SPC\text{-}dev_{i,j} \leq \lceil 2/r_{i,j} \rceil,\ spc\text{-}dev_{i,j} \leq \lceil 1/r_{i,j} \rceil.$$

Theorem 4.5: *In a sBUX switch, once a cell of flow $f_{i,j}$ goes to the head of $VOQ_{i,j}$, it will wait at most $\lfloor 3/r_{i,j} \rfloor$ slots before it leaves the crosspoint buffer $XPB_{i,j}$ (excluding the constant fabric-internal propagation latency common to all cells).*

Simulation shows that all the bounds of Theorems 4.1–4.5 are tight in the sense that they are achievable at certain instances. For example, observe the instance where $N = 3$, $i = 3$, $j = 2$, $r_{i,j} = 3/15$, row vector $= (1/15, 3/15, 11/15)$, and column vector $= (2/15, 10/15, 3/15)$. Starting from $I_{i,j} = 0$, the second output operation on $XPB_{i,j}$ at slot 7 is ineffective, and $XPB_{i,j}$ occupancy reaches 2 at the end of slot 12. In the cell sequence out of output j, the first cell coming from input i occurs at slot 1, the second at slot 13. This produces an interval of length 11 with no cell service to input i covered, or a covering deviation of 2 cell services. This also produces a 1-step space of 12 slots, or a spacing deviation of 7 slots, more than $\lceil 1/r_{i,j} \rceil = 5$ but less than $\lceil 2/r_{i,j} \rceil = 10$ slots. After the first cell of flow $f_{i,j}$ is scheduled into $XPB_{i,j}$ at slot 1, the second cell becomes the head of $VOQ_{i,j}$ and waits 12 slots before it departs from $XPB_{i,j}$; the waiting time is more than $\lfloor 2/r_{i,j} \rfloor = 10$ but less than $\lfloor 3/r_{i,j} \rfloor = 15$.

In general, given an admissible bandwidth matrix, if the switch, possibly with constant speedup support, can guarantee constant smoothness bounds $cvr\text{-}dev_{i,j}$ and $CVR\text{-}dev_{i,j}$ for each flow from input i to output j, then we define that this switch achieves *smooth switching*. Actually, such constant smoothness bounds imply that the switch can use 100% of the switch capacity to provide deterministic rate guarantees for each flow with constant fairness and jitter bounds, or perfect predictability. By definition, sBUX achieves smooth switching.

D. Summary

When supporting real-time services, the CICQ switch sBUX best achieves the two goals of scalability and predictability. sBUX requires the same memory bandwidth as an IQ switch without speedup, which is the lowest possible. Its computation is based on rate rather than per-slot information, and is conducted in a fully distributed manner; this minimizes communication overhead. The key operation of its computation, namely the minimum selection among multiple numbers, is amenable to fast hardware implementation, and seems indispensable to any scheduler with guaranteed performance under non-uniform traffic. It eliminates credit-based feedback flow control for the crosspoint buffer, decreases the crosspoint buffer size to the minimal, eliminates output buffer since no speedup is needed; all these greatly simplify the implementation. Its guaranteed smoothness bounds under any admissible bandwidth matrix are small constants, implying almost the best achievable qualities of service including throughput, bandwidth and fairness, delay and jitter guarantees. In particular, these guarantees are deterministic, worst-case, and finite-scaled guarantees, rather than stochastic, average-case, and asymptotic ones. To our best knowledge, no other CICQ or CIOQ (combined input- and output-queued) switches have been proved to hold such comprehensive quality of service guarantees. In short, sBUX is a transparent switch.

V. Bandwidth Regulator

sBUX is ready to support bandwidth-guaranteed services since an admissible bandwidth matrix is already available through admission control. When sBUX is to support best-effort services, such an admissible bandwidth matrix is not available, and has to be obtained dynamically via a bandwidth regulator, whose design is described in this section.

A. Architecture

Fig. 3 is the architecture of the bandwidth regulator, which is composed of distributed bandwidth demand estimators and a centralized bandwidth allocator. *Periodically*, bandwidth demand estimators collect information about traffic arrivals and VOQ backlogs at each input port, and produce an on-line estimation of the bandwidth demand matrix. The bandwidth allocator receives the possibly *inadmissible* bandwidth demand matrix, transforms it into an *admissible* and *fully loaded* one, and then feeds it to the smoothed switch sBUX for scheduling in the next period.

B. Bandwidth Demand Estimator

Two sources of information can be used to design a bandwidth demand estimator: traffic arrival and VOQ backlog. To compensate for the one period lag due to bandwidth allocation computing, bandwidth demand estimation at the beginning of a period is to estimate the demand in the next period instead of the current immediate one.

Traffic arrival prediction receives much attention by previous research, which uses the common technique of exponentially weighted moving average filter to predict the future arrival based on the information of past (factual) arrival and current (predicted) one [4], [5]. Specifically, at the beginning of the kth period, we are to predict the arrival rate $r^p_{i,j}(k+1)$ of flow $f_{i,j}$ in the $(k+1)$th period, based on the factual arrival rate $r^f_{i,j}(k-1)$ in the $(k-1)$th period and the predicted arrival rate $r^p_{i,j}(k)$ in the kth period:

$$r^p_{i,j}(k+1) = \alpha \cdot r^f_{i,j}(k-1) + (1-\alpha) \cdot r^p_{i,j}(k)$$

where α is the gain or forgetting factor, $0 < \alpha \leq 1$.

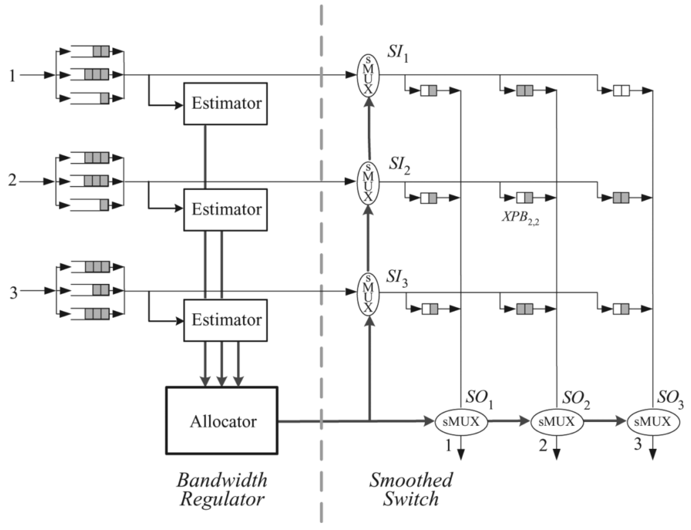


Fig. 3. Architecture of the bandwidth regulator.

$$\begin{bmatrix}4&5&6\\3&5&5\\2&1&6\end{bmatrix} \rightarrow \begin{bmatrix}2&3&3\\2&3&2\\2&0&3\end{bmatrix} \rightarrow \begin{bmatrix}\mathbf{4}&3&3\\2&\mathbf{6}&2\\2&0&\mathbf{5}\end{bmatrix} \rightarrow \begin{bmatrix}4&\mathbf{3}&3\\2&6&\mathbf{2}\\\mathbf{4}&0&5\end{bmatrix} \rightarrow \begin{bmatrix}4&3&\mathbf{3}\\\mathbf{2}&6&2\\4&\mathbf{1}&5\end{bmatrix}$$

Fig. 4. Illustration of bandwidth allocation. The first is the bandwidth demand matrix. The second is the bandwidth allocation matrix after proportional scaling. The third to fifth are the results of three steps of the bandwidth booster.

The VOQ backlog information reflects both the actual arrival and the actual allocation information; it is the deficit of allocation compared with arrival accumulated in the past. This information is actually more important since, based solely on it, a maximum weight matching scheduler *longest queue first* guarantees 100% throughput for a crossbar switch [12], [28]. Specifically, at the beginning of the kth period, we are to predict the backlog $q^p_{i,j}(k+1)$ of flow $f_{i,j}$ at the beginning of the $(k+1)$th period. While previous research uses $q^p_{i,j}(k+1) = q^f_{i,j}(k)$, we propose a more accurate design of estimator $q^p_{i,j}(k+1)$ by taking into account the current factual backlog $q^f_{i,j}(k)$, plus the arrival rate $r^p_{i,j}(k)$ and the allocated rate $r^a_{i,j}(k)$ for the current period:

$$q^p_{i,j}(k+1) = max\left\{0, q^f_{i,j}(k) + \left(r^p_{i,j}(k) - r^a_{i,j}(k)\right) \cdot T\right\}$$

where T is the time of period.

The final bandwidth demand estimator $r^d_{i,j}(k)$ we use is the sum of the arrival and the backlog estimators:

$$r^d_{i,j}(k+1) = r^p_{i,j}(k+1) + q^p_{i,j}(k+1)/T.$$

It is worth noting that certain demand estimators do not fully take backlog information into account, which simplify the following bandwidth allocation but probably at the cost of dynamic response [4], [5].

C. Bandwidth Allocator

The design of bandwidth allocator should take complexity, throughput and fairness into account. Previous research considers the maximum flow model best for this purpose, with bandwidth demand and line rate as capacity constraints [4], [5]. However, if the period degenerates into one slot, i.e., $T = 1$, then the maximum flow degenerates into a maximum size matching, which does not guarantee 100% throughput. Besides, the maximum flow computation is of high complexity. Therefore, this maximum flow model is neither sound in principle nor feasible in practice.

Another choice is the max-min fair allocator [17]. It is an extension of the classical one-dimensional max-min fair allocator to the two-dimensional matrix, and guarantees that the minimum allocation to an element is the maximum among all admissible allocations. But its computing complexity is also high, and it is more fairness-oriented than throughput-oriented.

Instead, we use the proportional scaler, the simplest among the three. It is amenable to fast hardware implementation and provides proportional fairness. Specifically, suppose flow $f_{i,j}$ demands bandwidth $r^d_{i,j}$, with total demand $r^d_{i,+} = \Sigma_t r^d_{i,t}$ at row or input i, and total demand $r^d_{+,j} = \Sigma_s r^d_{s,j}$ at column or output j. Then the proportional scaler makes an allocation $r^a_{i,j} = r^d_{i,j}/max\{r^d_{i,+}, r^d_{+,j}\}$. Such an allocation is admissible, because $\Sigma_j r^a_{i,j} \leq \Sigma_j r^d_{i,j}/r^d_{i,+} = 1$, and $\Sigma_i r^a_{i,j} \leq \Sigma_i r^d_{i,j}/r^d_{+,j} = 1$.

Under rate-based switch scheduling with periodic rate updating, an admissible yet not fully loaded bandwidth allocation matrix will waste the excess capacity of the sBUX switch. The proportional scaler has a unique merit that is not available with the maximum flow and the max-min fair allocators. That is, *iterated* proportional scaling can boost bandwidth allocation proportionally while keeping admissibility, and guarantee to converge, possibly to a doubly stochastic or fully loaded matrix if each element is originally positive. However, within a *fixed* number of iterations, proportional allocation cannot converge to a doubly stochastic matrix; there is still excess bandwidth to be used. Therefore, after applying proportional scaler just once, we use a bandwidth booster to obtain a doubly stochastic allocation matrix.

The bandwidth booster is an extension of the Inukai algorithm for the same purpose. The Inukai algorithm starts from position (1, 1), boosts the element to saturate at least one of the associated two lines, then makes a down or right move and repeats boosting, until a doubly stochastic matrix is obtained within $(2N - 1)$ steps [15]. This algorithm is unfair since elements appearing earlier in the path get more chances of boosting; besides, the algorithm is sequential. We propose a parallel and fair booster. Specifically, partition the matrix into N diagonals, each one corresponding to a perfect match; boost in parallel the N elements in each diagonal to saturation, which is fair to rows and columns; finish N diagonals in N steps. To further improve fairness, in each period, we select a new starting diagonal in the round-robin manner.

Fig. 4 is an illustration of how to transform a bandwidth demand matrix into a doubly stochastic one by the proportional scaler and the bandwidth booster. The matrices have been transformed into integral ones, with period $T = 10$ as the target line sum. The proportional scaler works as $r^a_{i,j} \leftarrow \lfloor r^d_{i,j} \times T/max\{r^d_{i,+}, r^d_{+,j}\}\rfloor$. The bandwidth booster works as $r^a_{i,j} \leftarrow r^a_{i,j} + T - max\{r^a_{i,+}, r^a_{+,j}\}$. Such integral representation and operation are amenable to hardware implementation.

D. Hardware Implementation

The implementation efficiency of the bandwidth regulator determines the minimum period T that is affordable in dynamic bandwidth updating.

TABLE II
TIME TO IMPLEMENT THE BANDWIDTH REGULATOR

N	T	*OC*-192	*OC*-768
16	1 µs	24 slots	97 slots
32	3 µs	50 slots	200 slots
64	5 µs	106 slots	423 slots
128	12 µs	235 slots	942 slots
256	29 µs	567 slots	2268 slots
512	78 µs	1518 slots	6072 slots
1024	234 µs	4572 slots	18288 slots
$T = 2N{\cdot}L/R + N^2{\cdot}(t_c + t_d)/M + (6t_a + t_c){\cdot}N$			
L = 16 b, R = 800 Mbps, M = 100, $t_c = t_a$ = 5 ns, t_d = 10 ns, cell = 64 B			

The bandwidth demand estimators are distributed in line cards, and their implementation is trivial. Each line card will transmit N bandwidth demands to the bandwidth allocator in the switch core, and then receive N bandwidth allocations from the allocator. All line cards work in parallel. Assume each bandwidth demand or allocation is encoded as an L-bit integer, and is transmitted between line cards and the switch core through high-speed serial links at speed R, then the total time of transmission is $2N \cdot L/R$.

The proportional scaling is done by $r^a_{i,j} \leftarrow \lfloor r^d_{i,j} \times T/max\{r^d_{i,+}, r^d_{+,j}\}\rfloor$. The two line sums $r^d_{i,+}$ and $r^d_{+,j}$ can be computed along with the receiving of bandwidth demands in a pipelined manner, and hence no additional time cost. Each comparison operation takes time t_c. The multiplication by T takes no time since T can be set as a power of two. Each division operation takes time t_d. Therefore, the proportional scaling of N^2 elements needs time $N^2 \cdot (t_c + t_d)$. Since all these operations are independent, they can be done in parallel with M parallel components, which reduces the time to $N^2 \cdot (t_c + t_d)/M$.

With a proportional bandwidth allocation $r^a_{i,j}$ in hand, before boosting, we should first compute the $2N$ line sums $r^a_{i,+}$ and $r^a_{+,j}$. Each element $r^a_{i,j}$ is involved in two additions, and elements in each of the N diagonals can be computed in parallel. This incurs time $2N^2 \cdot t_a/N = 2N \cdot t_a$, where t_a is the time of addition operation.

Then the bandwidth boosting $r^a_{i,j} \leftarrow r^a_{i,j} + T - max\{r^a_{i,+}, r^a_{+,j}\}$ is done in N steps. At each step, N elements in a diagonal are boosted in parallel. Each boosting operation adds $T - max\{r^a_{i,+}, r^a_{+,j}\}$ to $r^a_{i,j}$, $r^a_{i,+}$, and $r^a_{+,j}$, and involves one comparison and four addition operations. The N steps need time $(4t_a + t_c) \cdot N$.

To sum up, the total time for the bandwidth regulation is $2N \cdot L/R + N^2 \cdot (t_c + t_d)/M + 2N \cdot t_a + (4t_a + t_c) \cdot N$. With available hardware technology, $L = 16$ b, $R = 800$ Mbps, $t_c = t_a = 5$ ns, $t_d = 10$ ns, $M = 100$. We conduct emulation with FPGA chip EP2S180F1508C3 (Altera Stratix II). Since each bandwidth demand is coded in $L = 16$ bits, for $N \leq 1024$, the line sum can be coded in 26 bits. By 20-stage pipelining, a 26-bit divider can be implemented in 10 ns. And 100 dividers consume only 61% of the chip resource. Table II presents the time estimation for different switch size N, in units of μs and slot of 64-byte cell at 10 Gb/s (OC-192) and 40 Gb/s (OC-768) link speed. Even for $N = 1024$, 16-bit encoding is enough to represent the number of cells arrived in a period of bandwidth regulation. For $N \leq 64$, rate information can be extracted within several microseconds.

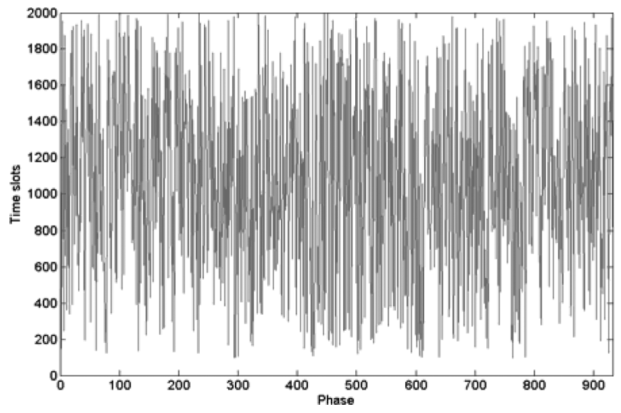


Fig. 5. The durations of 932 phases in 1 000 000 slots, each generated randomly in [100, 2000].

VI. Simulation

In this section, we evaluate the performance of sBUX plus bandwidth regulator under best-effort traffic.

A. Simulation Design

There are three key components for the simulation design: switch settings, traffic patterns, and performance measures.

The switch is the integration of the bandwidth regulator and the smoothed buffered crossbar sBUX, with switch size $N = 8$, 16, 32, 64, and bandwidth regulation period $T = 32$, 64, 128, 256, 512, 1024, 2048, 4096.

At inputs, random linear combinations of three key traffic patterns with random durations are generated to reflect the dynamic nature of arriving traffic. This is called the synthetic pattern, more difficult to deal with than its individual component. The three component traffic patterns are the uniform (U), log-diagonal (LD), and unbalanced (UB) [38]; the last one is used at unbalanced factor $w = 0.5$ as that is harsh to most switches. The simulation lasts 1 000 000 slots and contains 932 phases whose durations are generated randomly within [100, 2000] (Fig. 5). In each phase, three random coefficients ϕ_1, ϕ_2 and ϕ_3 $(\phi_1 + \phi_2 + \phi_3 = 1)$ are generated, and $(\phi_1 \cdot \text{U} + \phi_2 \cdot \text{LD} + \phi_3 \cdot \text{UB}_{0.5})$ is synthesized as the traffic pattern for this phase. Traffic load ρ varies from 0.1 to 1.0 with an increment of 0.1 when $0.1 \leq \rho \leq 0.8$ and an increment of 0.05 when $0.8 \leq \rho \leq 1.0$. We also use a bursty traffic pattern, i.e., an on-off arrival process modulated by a two-state Markov chain to test the effect of burstiness on sBUX [27].

At outputs, throughput, average delay and burstiness are measured over the whole simulation duration of 1 000 000 slots. Throughput is the ratio of the number of cells that appear at outputs to the number of cells that arrive at inputs, when each input is fully loaded $(\rho = 1.0)$. Average delay is measured over all cells that appear at outputs and under all loads $0.1 \leq \rho \leq 1.0$. Average burstiness is measured over all the segments at outputs, with each segment being a maximal subsequence of consecutive cells from the same input.

B. Simulation Results

Simulation results are presented in three parts.

The first part studies the bandwidth demand estimator design. Besides selecting α, we also test the bandwidth demand estimator that is based solely on backlog, just for comparison. Sim-

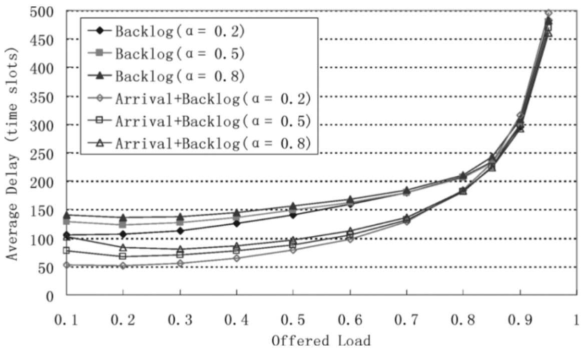


Fig. 6. Average delay under synthetic traffic at $N = 32, T = 256$.

TABLE III
THROUGHPUT UNDER SYNTHETIC TRAFFIC

N \ T	32	64	128	256	512	1024	2048	4096
8	.999	.999	.998	.998	.998	.998	.997	.996
16	.998	.998	.998	.998	.997	.997	.996	.995
32	.995	.998	.998	.997	.997	.996	.996	.994
64	.986	.994	.995	.995	.995	.994	.994	.993

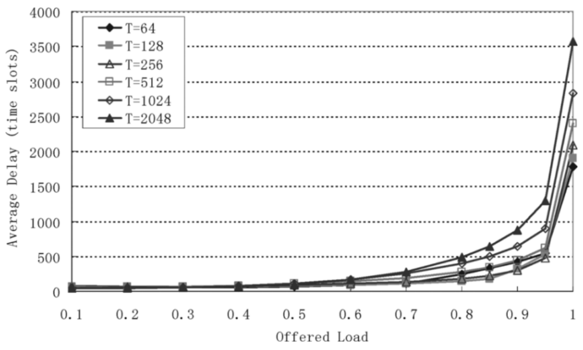


Fig. 7. Average delay under synthetic traffic at $N = 32$.

ulation is done under synthetic traffic at $N = 32$, $T = 256$, with α varied in [0.1, 1.0] at an increment of 0.1. All throughputs are 0.997, no difference at all; but average delays differ, as Fig. 6 shows. Specifically, by addition of arrival prediction, average delay can be reduced by at least 35% at medium load ($\rho = 0.50$). The choice of α, however, seems insignificant. Simulations with other settings of N and T produce similar results. To keep balance between past and present, we select $\alpha = 0.5$ in subsequent simulations. This choice simplifies hardware implementation of the arrival predictor.

The second part studies the effects of period T and switch size N. All combinations of switch size $N = 8$, 16, 32, 64, and period $T = 32$, 64, 128, 256, 512, 1024, 2048, 4096 are tested. Table III shows the throughputs, all of which are almost 100%, with marginal difference between each other.

Fig. 7 shows the average delay under synthetic traffic at varied periods (T) yet unvaried switch size ($N = 32$). When T increases, the average delay increases slowly. The tendency is the same with other N. With $N = 32$, $T = 256$, 64-byte cell and 40-Gb/s link speed, the average delay is 470 slots or 6 μs at $\rho = 0.95$.

Fig. 8 shows the average delay under synthetic traffic at varied switch size (N) yet unvaried period ($T = 256$). When N increases, the average delay increases slowly. The tendency is the same with other T.

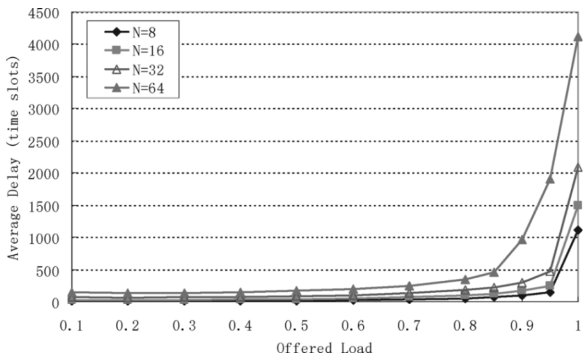


Fig. 8. Average delay under synthetic traffic at $T = 256$.

TABLE IV
PERFORMANCES UNDER BURSTY TRAFFIC ($N = 32, T = 256$)

	Burst length = 8 (throughput = .994)		Burst length = 16 (throughput = .993)		Burst length = 32 (throughput = .991)	
ρ	Avg. burst	Avg. delay	Avg. burst	Avg. delay	Avg. burst	Avg. delay
0.60	1.0	394	1.0	466	1.0	534
0.70	1.0	437	1.0	518	1.0	603
0.80	1.0	505	1.0	603	1.0	729
0.85	1.0	561	1.0	683	1.0	880
0.90	1.0	667	1.0	873	1.0	1235
0.95	1.0	998	1.0	1406	1.0	2069
1.00	1.0	4171	1.0	5289	1.0	7073

The third part studies the performance under bursty traffic. As Table IV shows, when input burst length increases, average delay also increases, but output burst length remains 1 for any loading, light or heavy, meaning perfect burstiness reduction by sBUX. The throughput remains over 99%.

VII. CONCLUSION

Rate-based smoothed switching is the design approach we propose in this paper, which decomposes a switch design into a generic bandwidth regulator and a specific smoothed switch, with an aim to achieve greater scalability and predictability.

Rate-based control is in sharp contrast to the predominant approach of credit-based control. The latter uses instantaneous buffer occupancy information at each time slot to compute the schedule and prevent buffer overflow. However, this per-slot computing and communication mode is difficult to scale with faster link speed and larger switch size. The scalability consideration urges us to use more compact information and coarse-grained or batch mode of processing. Rate information is compact in nature as it is applicable for a period of time slots, and can be extracted feasibly in microseconds by our bandwidth regulator. The rate-based smoothed buffered crossbar sBUX removes the credit-based flow control, augments the distributed feature of input and output scheduling, reduces the capacity of each crosspoint buffer to a minimal constant of two cells, and eliminates the need of speedup. All these greatly simplify the implementation and enhance the scalability.

Rate-based smoothed switching is also highly predicable. With just a two-cell buffer at each crosspoint and no speedup, sBUX guarantees 100% throughput for real-time services. In particular, the almost ideal smoothness guarantees imply that rate guarantees are very accurate even in the worst case and in any time window, short or long, finite or infinite. In some sense, the switch is so predictable that it can be considered transparent. Simulation shows that coupled with bandwidth regulator, the switch guarantees almost 100% throughput and perfect burstiness reduction for best-effort services, with average delays of at most tens of microseconds even under heavy load ($\rho = 0.95$) and reasonably large switch size ($N \le 64$). It is worth noting that the approach of rate-based smoothed switching can support best-effort and real-time services gracefully in the same model, an obvious yet important feature we do not elaborate in the paper.

One of the key problems left is to design a bandwidth regulator with guaranteed 100% throughput and with improved average delay and implementation efficiency. We conjecture that based on just queueing backlog information, the proportional allocator, after sufficient iterations, can guarantee 100% throughput for best-effort services.

The smooth switching problem offers a critical perspective for switch design and analysis, and remains open for most switches, even for the classical single-stage crossbar switch despite the feasible perfect emulations of certain push-in, first-output OQ switches [10], [11]. We conjecture that a crossbar switch can achieve smooth switching with a speedup of no more than two. Considering that even rate guarantees are difficult to obtain for most switches, the smoothness guarantee may be a luxury. But smoothness does play an indispensable role in bounding the buffer usage in the switches, as demonstrated by sBUX.

The further scalability of buffered crossbar switches might suffer from the N^2 crosspoints. When the switch size scales to hundreds or even thousands, the three-stage buffered crossbar switches are preferable; a recent example is a 1024 × 1024 three-stage switch, with thirty-two 32 × 32 buffered crossbar chips at each stage [9]. There is a solution for rate-based three-stage *bufferless* crossbar switches [5], but we hope that with sBUX as switching element, the rate-based three-stage *buffered* switch fabrics can be made simpler in implementation and more predictable in performance.

APPENDIX

A. Proof of Theorem 2.1

The theorem is correct when the interval for definition contains exactly one cell service to flow f_i. Let us consider the general cases where the interval for definition contains at least two cell services to flow f_i. The proof is by contradiction.

(1) Assume that $cvr_i(l) \ge m$, but $SPC_i(m) > l$. Then $\exists j_0$, such that $SPC_i(m) = Pos_i(j_0 + m) - Pos_i(j_0) > l$. Hence, $\exists t' = Pos_i(j_0) + 1$, $l' = SPC_i(m) - 1 \ge l$, such that $Cover_i(t', l') = m - 1 < m$. Therefore $cvr_i(l) \le Cover_i(t', l) \le Cover_i(t', l') < m$, contradictory with the assumption $cvr_i(l) \ge m$.

(2) Assume that $SPC_i(m) \le l$, but $cvr_i(l) < m$. Then $\exists t'$, such that $cvr_i(l) = Cover_i(t', l) \le m - 1$. Assume that $SPC_i(m)$ is defined within $L = [Pos_i(j_1), Pos_i(j_2)+1)$, then $j_1 + m \le j_2$. Since $cvr_i(l)$ is also defined within L, hence $Pos_i(j_1) \le t' \le t' + l - 1 \le Pos_i(j_2)$.

If $Pos_i(j_1) = t'$, since $Cover_i(t', l) \le m - 1$, it is implied that $t'+l+1 \le Pos_i(j_1+m) \le Pos_i(j_2)$. Hence, $SPC_i(m) \ge Space_i(j_1, m) \ge l + 1$, contradiction. It is the same with $t' + l - 1 = Pos_i(j_2)$.

If $Pos_i(j_1) < t' \le t' + l - 1 < Pos_i(j_2)$, we can locate the cell service to flow f_i immediately *before* slot t' at $Pos_i(j_3) \in [Pos_i(j_1), t')$ and locate the cell service to flow f_i immediately *after* slot $(t' + l - 1)$ at $Pos_i(j_4) \in [t' + l, Pos_i(j_2) + 1)$. Since $Cover_i(t', l) \le m - 1$, then $j_4 - j_3 \le m$, and $SPC_i(m) \ge Pos_i(j_4) - Pos_i(j_3) \ge l + 1$, contradiction.

(3) Assume that $CVR_i(d) \le s$, but $spc_i(s) < d$. Then $\exists j_0$, such that $spc_i(s) = Pos_i(j_0 + s) - Pos_i(j_0) < d$. The interval $L = [Pos_i(j_0), Pos_i(j_0 + s) + 1)$ contains $(s + 1)$ number of cell services to flow f_i, but has a length of at most $l = (spc_i(s) + 1) \le d$. Then $CVR_i(d) \ge CVR_i(l) \ge Cover_i(Pos_i(j_0), l) = s + 1$, contradictory with the assumption $CVR_i(d) \le s$.

(4) Assume that $spc_i(s) \ge d$, but $CVR_i(d) > s$. Then $\exists t'$, such that $Cover_i(t', d) = CVR_i(d) \ge s + 1$. Within interval $[t', t' + d)$, there are at least $(s + 1)$ number of cell services to flow f_i, which decide an s-step space of at most $(d - 1)$, contradictory with the assumption $spc_i(s) \ge d$. □

B. Proof of Theorem 2.2

The theorem is correct when the interval for definition contains exactly one cell service to flow f_i. Let us consider the general cases where the interval for definition contains at least two cell services to flow f_i. We will use two general properties concerning any integer p and real number x:

Property 1: $x - 1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x + 1$.

Property 2: $p < x + 1 \Leftrightarrow p \le \lceil x \rceil$.

(1)~(2) Assume that $cvr\text{-}dev_i$ and $CVR\text{-}dev_i$ are defined within interval $[t_1, t_1 + l_1)$, which covers totally $(k_1 + 1)(k_1 \ge 1)$ number of cell services to flow f_i, with the first at slot $Pos_i(j_1)$ and the last at slot $Pos_i(j_1 + k_1)$; $SPC\text{-}dev_i$ and $spc\text{-}dev_i$ are defined within $[Pos_i(j_1), Pos_i(j_1 + k_1) + 1) \subseteq [t_1, t_1 + l_1)$.

Define $cvr'_i(l) = min\{Cover_i(t_2, l) | [t_2, t_2 + l) \subseteq [Pos_i(j_1), Pos_i(j_1 + k_1) + 1)\}$, i.e., $cvr'_i(l)$ is defined within the interval for definition of spacing properties $SPC\text{-}dev_i$ and $spc\text{-}dev_i$. Since the domain of definition of $cvr'_i(l)$ is a subset of that of $cvr_i(l)$, $cvr'_i(l) \ge cvr_i(l)$. In the same way we define $CVR'_i(l)$, and $CVR'_i(l) \le CVR_i(l)$.

(1) For any step size s, $1 \le s \le k_1$, let $SPC_i(s) = l_s$, then $cvr'_i(l_s - 1) = s - 1$ (since $SPC_i(s) \le l_s$, by Theorem 2.1, $cvr'_i(l_s) \ge s$, hence $cvr'_i(l_s - 1) \ge s - 1$. If $cvr'_i(l_s - 1) \ge s$, by Theorem 2.1, $SPC_i(s) \le l_s - 1$, contradiction).

$$\begin{aligned} SPC_i(s) - \lceil s/w_i \rceil &\le l_s - s/w_i \\ &= ((l_s - 1) \cdot w_i - 1 - (s - 1))/w_i + 1 \\ &< (\lfloor (l_s - 1) \cdot w_i \rfloor - cvr'_i(l_s - 1))/w_i + 1 \\ &\qquad \text{(by Property 1).} \end{aligned}$$

By Property 2,

$$\begin{aligned} SPC_i(s) - \lceil s/w_i \rceil &\le \lceil (\lfloor (l_s - 1) \cdot w_i \rfloor - cvr'_i(l_s - 1))/w_i \rceil \\ &\le \lceil (\lfloor (l_s - 1) \cdot w_i \rfloor - cvr_i(l_s - 1))/w_i \rceil \\ &\le \lceil cvr\text{-}dev_i / w_i \rceil. \end{aligned}$$

Hence, $SPC\text{-}dev_i \le \lceil cvr\text{-}dev_i / w_i \rceil$.

(2) For any step size s, $1 \le s \le k_1$, let $spc_i(s) = l_s$, then $CVR'_i(l_s + 1) = s + 1$ (since $spc_i(s) \ge l_s$, by Theorem 2.1,

$CVR'_i(l_s) \le s$, hence $CVR'_i(l_s+1) \le s+1$; if $CVR'_i(l_s+1) \le s$, by Theorem 2.1, $spc_i(s) \ge l_s + 1$, contradiction).

$$\begin{aligned}\lfloor s/w_i \rfloor - spc_i(s) &\le s/w_i - l_s \\ &= ((s+1) - ((l_s+1)\cdot w_i + 1))/w_i + 1 \\ &< (CVR'_i(l_s+1) - \lceil (l_s+1)\cdot w_i \rceil)/w_i + 1 \\ &\quad \text{(by Property 1)}.\end{aligned}$$

By Property 2,

$$\begin{aligned}\lfloor s/w_i \rfloor - spc_i(s) &\le \lceil (CVR'_i(l_s+1) - \lceil (l_s+1)\cdot w_i \rceil)/w_i \rceil \\ &\le \lceil (CVR_i(l_s+1) - \lceil (l_s+1)\cdot w_i \rceil)/w_i \rceil \\ &\le \lceil CVR\text{-}dev_i/w_i \rceil.\end{aligned}$$

Hence, $spc\text{-}dev_i \le \lceil CVR\text{-}dev_i/w_i \rceil$.

(3)~(4) Assume that $SPC\text{-}dev_i$ and $spc\text{-}dev_i$ are defined within interval $[Pos_i(j_1), Pos_i(j_1+k_1)+1)$, $k_1 \ge 1$, which has a length of $l' = Space_i(j_1, k_1) + 1$. The $cvr\text{-}dev_i$ and $CVR\text{-}dev_i$ are defined within the same interval.

(3) For any interval length l, $1 \le l \le l'$, let $cvr_i(l) = s_l$.

If $1 \le l \le l'-2$, then $s_l \le k_1 - 1$, and $SPC_i(s_l+1) \ge l+1$ (if $SPC_i(s_l+1) \le l$, then by Theorem 2.1, $cvr_i(l) \ge s_l + 1$, contradiction).

$$\begin{aligned}\lfloor l\cdot w_i \rfloor - cvr_i(l) &\le l\cdot w_i - s_l \\ &= (l+1-((s_l+1)/w_i+1))\cdot w_i + 1 \\ &< (l+1-\lceil (s_l+1)/w_i \rceil)\cdot w_i + 1 \\ &\quad \text{(by Property 1)} \\ &\le (SPC_i(s_l+1) - \lceil (s_l+1)/w_i \rceil)\cdot w_i + 1 \\ &\le SPC\text{-}dev_i \cdot w_i + 1.\end{aligned}$$

By Property 2, $\lfloor l\cdot w_i \rfloor - cvr_i(l) \le \lceil SPC\text{-}dev_i \cdot w_i \rceil$.

If l' and k_1 are finite, we have to deal with two special cases. Note that $cvr_i(l'-1) = k_1$, $cvr_i(l') = k_1 + 1$, $SPC_i(k_1) = l' - 1$.

If $l = l' - 1$,

$$\begin{aligned}&\lfloor (l'-1)\cdot w_i \rfloor - cvr_i(l'-1) \\ &\quad \le (l' - 1 - (k_1/w_i + 1))\cdot w_i + 1 \\ &\quad < (SPC_i(k_1) - \lceil k_1/w_i \rceil)\cdot w_i + 1 \quad \text{(by Property 1)} \\ &\quad \le SPC\text{-}dev_i \cdot w_i + 1.\end{aligned}$$

By Property 2,

$$\lfloor (l'-1)\cdot w_i \rfloor - cvr_i(l'-1) \le \lceil SPC\text{-}dev_i \cdot w_i \rceil.$$

If $l = l'$,

$$\begin{aligned}\lfloor l' \cdot w_i \rfloor - cvr_i(l') &= \lfloor l' \cdot w_i \rfloor - 1 - k_1 \\ &\le \lfloor (l'-1)\cdot w_i \rfloor - cvr_i(l'-1) \\ &\le \lceil SPC\text{-}dev_i \cdot w_i \rceil.\end{aligned}$$

Therefore, $cvr\text{-}dev_i \le \lceil SPC\text{-}dev_i \cdot w_i \rceil$.

(4) For any interval length l, $1 \le l \le l'$, let $CVR_i(l) = s_l$, then $spc_i(s_l - 1) \le l - 1$ (if $spc_i(s_l-1) \ge l$, then by Theorem 2.1, $CVR_i(l) \le s_l - 1$, contradiction).

$$\begin{aligned}CVR_i(l) - \lceil l\cdot w_i \rceil &\le s_l - l\cdot w_i = ((s_l-1)/w_i - l)\cdot w_i + 1 \\ &< (\lfloor (s_l-1)/w_i \rfloor - (l-1))\cdot w_i + 1 \\ &\quad \text{(by Property 1)} \\ &\le (\lfloor (s_l-1)/w_i \rfloor - spc_i(s_l-1))\cdot w_i + 1 \\ &\le spc\text{-}dev_i \cdot w_i + 1.\end{aligned}$$

By Property 2, $CVR_i(l) - \lceil l \cdot w_i \rceil \le \lceil spc\text{-}dev_i \cdot w_i \rceil$.
Therefore, $CVR\text{-}dev_i \le \lceil spc\text{-}dev_i \cdot w_i \rceil$. □

C. Proof of Theorem 2.4

First we prove the distribution of cell services to flow f_1 is ideal over arbitrary intervals. Consider arbitrary interval $[Pos_1(j), Pos_1(j+s)+1)$,

$$\begin{aligned}Space_1(j,s) &= \lceil x + (j+s)/w_1 \rceil - \lceil x + j/w_1 \rceil \\ &\Rightarrow \lfloor s/w_1 \rfloor \le Space_1(j,s) \le \lceil s/w_1 \rceil \\ &\Rightarrow spc\text{-}dev_1 = SPC\text{-}dev_1 = 0 \\ &\Rightarrow cvr\text{-}dev_1 = CVR\text{-}dev_1 = 0.\end{aligned}$$

Since j and s are arbitrary, $cvr\text{-}dev_1 = CVR\text{-}dev_1 = 0$ is valid for any interval.

Next, we prove the distribution of cell services to flow f_2 in the schedule is also ideal. For any slot t and interval length l, since the distribution of cell services to flow f_1 is ideal,

$$\lfloor l\cdot w_1 \rfloor \le Cover_1(t,l) \le \lceil l \cdot w_1 \rceil.$$

Since $Cover_1(t,l) + Cover_2(t,l) = l$, $w_1 + w_2 = 1$, we have

$$\lfloor l\cdot w_2 \rfloor \le Cover_2(t,l) \le \lceil l \cdot w_2 \rceil.$$

That is, $cvr\text{-}dev_2 = CVR\text{-}dev_2 = 0$.

Therefore, the distributions of cell services to flow f_1 and f_2 in the schedule are both ideal, so the schedule is ideal. □

D. Proof of Theorem 3.1

Lemma 3.1 (Spuri): [41] Given a set of real-time jobs $J = \{J_i\} = \{(e_i, c_i, d_i)|i\}$, where job J_i requires a service time of c_i after its eligible time e_i but before its deadline d_i. Given an interval of time $[t_1, t_2)$, define the *processor demand* of the job set on interval $[t_1, t_2)$ as $h[t_1, t_2) = \Sigma c_k$ subject to $t_1 \le e_k$ and $d_k \le t_2$. Then any job set J is feasibly scheduled by EDF on uni-processor if and only if its processor demand $h[t_1, t_2) \le t_2 - t_1$ on any interval $[t_1, t_2)$. □

The jth cell service to flow f_i can be modeled as a real-time job $J_{i,j} = (\lceil e_{i,j} \rceil, c_{i,j}, \lceil d_{i,j} \rceil)$, where $e_{i,j} = I_i + (j-1)/w_i$, $c_{i,j} = 1$, and $d_{i,j} = I_i + j/w_i$. Scheduler sMUX is to schedule the job set $\{J_{i,j} | j \ge 1, 1 \le i \le n\}$ by EDF on uniprocessor.

Given an integral interval $[t_1, t_2)$, assume that k_i number of cell services to flow f_i, say, from the $(h+1)$th cell service to the $(h+k_i)$th, are completely contained inside the interval $[t_1, t_2)$. That is,

$$t_1 \le \lceil I_i + h/w_i \rceil, \; \lceil I_i + (h+k_i)/w_i \rceil \le t_2.$$

By $x \le \lceil x \rceil < x + 1$,

$$\begin{aligned}&t_1 < I_i + h/w_i + 1, \; I_i + (h+k_i)/w_i \le t_2. \\ &k_i < (t_2 - t_1 + 1)\cdot w_i.\end{aligned}$$

The processor demand of job set $\{J_{i,j}\}$ on $[t_1, t_2)$ is

$$h[t_1, t_2) = \Sigma_i k_i < (t_2 - t_1 + 1)\cdot \Sigma_i w_i \le (t_2 - t_1 + 1).$$

That is, $\Sigma_i k_i < (t_2 - t_1 + 1)$. Since both sides of $<$ are integers, $\Sigma_i k_i \le t_2 - t_1$.

Given an arbitrary interval $[t_1, t_2)$, then

$$h[t_1, t_2) = h[\lceil t_1 \rceil, \lfloor t_2 \rfloor) \le \lfloor t_2 \rfloor - \lceil t_1 \rceil \le \lfloor t_2 - t_1 \rfloor \le t_2 - t_1.$$

Therefore, by Lemma 3.1, the job set is feasibly scheduled by sMUX. □

E. Proof of Theorem 3.2

(1) By Theorem 3.1,

$$\begin{aligned} Pos\,(S,\tau_i,\lfloor l\cdot w_i\rfloor) &\le \lceil I_i+\lfloor l\cdot w_i\rfloor/w_i\rceil-1\\ &\le I_i+l-1. \end{aligned}$$

That is, the $\lfloor l\cdot w_i\rfloor$th cell service to flow f_i is covered by interval $[I_i, I_i+l)$, therefore $\lfloor l\cdot w_i\rfloor \le Cover(S,\tau_i,I_i,l)$.

$$\begin{aligned} &Pos\,(S,\tau_i,(\lfloor (l-1)\cdot w_i\rfloor+1)+1)\\ &\quad\ge \lceil I_i+(\lfloor (l-1)\cdot w_i\rfloor+1)/w_i\rceil \quad \text{(Theorem 3.1)}\\ &\quad\ge I_i+(\lfloor (l-1)\cdot w_i\rfloor+1)/w_i\\ &\quad> I_i+((l-1)\cdot w_i)/w_i \quad (\lfloor x\rfloor+1>x)\\ &\quad= I_i+l-1, \quad \text{which is an integer; hence}\\ &Pos\,(S,\tau_i,(\lfloor (l-1)\cdot w_i\rfloor+1)+1)\ge I_i+l. \end{aligned}$$

This means the position of the $((\lfloor (l-1)\cdot w_i\rfloor+1)+1)$th cell service to flow f_i is beyond interval $[I_i, I_i+l)$. Therefore,

$$Cover(S,\tau_i,I_i,l)\le \lfloor (l-1)\cdot w_i\rfloor+1.$$

(2) For any $t\ge I_i$,

$$\begin{aligned} Cover(S,\tau_i,t,l) &= Cover(S,\tau_i,I_i,t-I_i+l)\\ &\quad - Cover(S,\tau_i,I_i,t-I_i)\\ &\ge \lfloor (t-I_i+l)\cdot w_i\rfloor\\ &\quad -(\lfloor (t-I_i-1)\cdot w_i\rfloor+1)\\ &\ge \lfloor (l+1)\cdot w_i\rfloor-1\\ &\qquad (\text{by relation } \lfloor x\rfloor-\lfloor y\rfloor\ge\lfloor x-y\rfloor);\\ Cover(S,\tau_i,t,l) &= Cover(S,\tau_i,I_i,t-I_i+l)\\ &\quad - Cover(S,\tau_i,I_i,t-I_i)\\ &\le (\lfloor (t-I_i+l-1)\cdot w_i\rfloor+1)\\ &\quad -(\lfloor (t-I_i)\cdot w_i\rfloor)\\ &\le \lceil (l-1)\cdot w_i\rceil+1\\ &\qquad (\text{by relation } \lfloor x\rfloor-\lfloor y\rfloor\le\lceil x-y\rceil). \end{aligned}$$

Therefore,

$$\lfloor (l+1)\cdot w_i\rfloor-1\le cvr_i(l)\le CVR_i(l)\le\lceil (l-1)\cdot w_i\rceil+1.$$

(3) Since $\lfloor (l+1)\cdot w_i\rfloor-1\le cvr_i(l)$, with an ideal schedule as reference, we obtain

$$cvr\text{-}dev_i\le max\,\{\lfloor l\cdot w_i\rfloor-(\lfloor (l+1)\cdot w_i\rfloor-1)\,|l\}\le 1.$$

In the same way, we can prove $CVR\text{-}dev_i\le 1$.

(4) By Theorem 3.1, in the sMUX schedule S, the jth $(j\ge 1)$ and the $(j+s)$th cell services to flow f_i satisfy

$$\begin{aligned} \lceil I_i+(j-1)/w_i\rceil &\le Pos(S,\tau_i,j)\le\lceil I_i+j/w_i\rceil-1,\\ \lceil I_i+(j+s-1)/w_i\rceil &\le Pos(S,\tau_i,j+s)\\ &\le\lceil I_i+(j+s)/w_i\rceil-1. \end{aligned}$$

Therefore, $\lfloor (s-1)/w_i\rfloor+1\le Space(S,\tau_i,j,s)\le\lceil (s+1)/w_i\rceil-1$, and $\lfloor (s-1)/w_i\rfloor+1\le spc_i(s)\le SPC_i(s)\le\lceil (s+1)/w_i\rceil-1$.

(5) Since $\lfloor (s-1)/w_i\rfloor+1\le spc_i(s)$, with an ideal schedule as reference, we have

$$\begin{aligned} spc\text{-}dev_i &\le max\,\{\lfloor s/w_i\rfloor-\lfloor (s-1)/w_i\rfloor-1|s\}\\ &\le\lceil 1/w_i\rceil-1\le\lfloor 1/w_i\rfloor, \text{ i.e.}\\ spc\text{-}dev_i &\le\lfloor 1/w_i\rfloor. \end{aligned}$$

In the same way we can prove $SPC\text{-}dev_i\le\lfloor 1/w_i\rfloor$. □

F. Proof of Theorem 4.1

$$\begin{aligned} ieCover(SO_j,\tau_{i,j},I_{i,j},l) &= max\,\{Cover(SO_j,\tau_{i,j},I_{i,j},l')\\ &\quad -eCover(SI_i,\tau_{i,j},I_{i,j},l')|l'=0,1,2,\ldots,l\}. \end{aligned}$$

Under the critical situation, all input operations are effective. By case (1) of Theorem 3.2,

$$\begin{aligned} Cover(SO_j,\tau_{ij},I_{ij},l') &\le \lfloor (l'-1)\cdot r_{i,j}\rfloor+1,\\ eCover(SI_i,\tau_{i,j},I_{i,j},l') &= Cover(SI_i,\tau_{i,j},I_{i,j},l').\\ &\ge \lfloor l'\cdot r_i,j\rfloor \end{aligned}$$

Therefore,

$$\begin{aligned} &Cover(SO_j,\tau_{i,j},I_{i,j},l')-eCover(SI_i,\tau_{i,j},I_{i,j},l')\\ &\qquad\le\lfloor (l'-1)\cdot r_{i,j}\rfloor+1-\lfloor l'\cdot r_{i,j}\rfloor\le 1. \end{aligned}$$

As a result, $ieCover(SO_j,\tau_{i,j},I_{i,j},l)\le 1$. □

G. Proof of Theorem 4.2

To obtain $XPBO_{i,j}$, it suffices to merely check the critical situation, under which all input operations are effective, and at most one output operation is ineffective (Theorem 4.1).

$$\begin{aligned} XPBO_{i,j} = max\,\{&eCover(SI_i,\tau_{i,j},I_{i,j},l)\\ &-Cover(SO_j,\tau_{i,j},I_{i,j},l)\\ &+ieCover(SO_j,\tau_{i,j},I_{i,j},l)|l=1,2,\ldots\}. \end{aligned}$$

By Theorem 3.2, case (1), and Theorem 4.1,

$$\begin{aligned} eCover(SI_i,\tau_{i,j},I_{i,j},l) &= Cover(SI_i,\tau_{i,j},I_{i,j},l)\\ &\le\lfloor (l-1)\cdot r_{i,j}\rfloor+1,\\ Cover(SO_j,\tau_{i,j},I_{i,j},l) &\ge\lfloor l\cdot r_{i,j}\rfloor,\\ ieCover(SO_j,\tau_{i,j},I_{i,j},l) &\le 1,\\ XPBO_{i,j} &\le max\{\lfloor (l-1)\cdot r_{i,j}\rfloor+1\\ &\quad -\lfloor l\cdot r_{i,j}\rfloor+1|l=1,2,\ldots\}\\ &\le 2. \end{aligned}$$

□

H. Proof of Theorem 4.3

Proof: The key to the proof is the fact that during each period of T slots, sMUX will conduct exactly $a_{i,j}$ number of input operations and just the same number of output operations. Again, to deduce the maximum occupancy of crosspoint buffer $XPB_{i,j}$, just consider the critical situation, in which all input operations are effective.

During the first period of T slots, Theorem 4.2 has proved

$$XPBO_{i,j} \leq 2.$$

In particular, the same number of input operations and output operations are conducted, and at most one output operation is ineffective (Theorem 4.1). Therefore, at the end of the first period, $XPB_{i,j}$ has at most one cell stored in it.

Now consider the second period. At the beginning, if there is no cell in $XPB_{i,j}$, then the conclusion is just the same as the first period.

If there is one cell stored in $XPB_{i,j}$ at the beginning, then there will be no ineffective output operation, because from the beginning of the period, the number of output operations exceeds the number of input operations by at most 1 (by case (1) of Theorem 3.2, $(\lfloor (l-1)\cdot r_{i,j}\rfloor + 1) - \lfloor l \cdot r_{i,j}\rfloor \leq 1$). Symmetrically, from the beginning of the period, the number of (effective) input operations exceeds the number of (effective) output operations by at most 1 (by the same reason as above). Taking the initial one cell into account, at any time during the second period, there are at most two cells in $XPB_{i,j}$, or $XPBO_{i,j} \leq 2$. Since the number of (effective) input operations and the number of (effective) output operations are equal, there will be one cell left in $XPB_{i,j}$ at the end of the second period.

The conclusion is the same with all the other periods. □

I. Proof of Theorem 4.4

(1) Assume that $VOQ_{i,j}$ is always backlogged. Let us derive the bounds to $eCover(SO_j, \tau_{i,j}, t, l)$, the number of effective output operations in interval $[t, t+l)$, $t \geq I_{i,j}$, with the switch core as our point of view.

For the upper bound, by Theorem 3.2,

$$\begin{aligned} eCover(SO_j, \tau_{i,j}, t, l) &\leq Cover(SO_j, \tau_{i,j}, t, l) \\ &\leq \lceil (l-1)\cdot r_{i,j}\rceil + 1 \\ &\leq \lceil l \cdot r_{i,j}\rceil + 1, \ i.e., \\ CVR\text{-}dev_{i,j} &\leq 1. \end{aligned}$$

For the lower bound, by Theorems 3.2 and 4.1,

$$\begin{aligned} eCover(SO_j, \tau_{i,j}, t, l) &= Cover(SO_j, \tau_{i,j}, t, l) \\ &\quad - ieCover(SO_j, \tau_{i,j}, t, l) \\ &\geq (\lfloor (l+1)\cdot r_{i,j}\rfloor - 1) - 1 \\ &\geq \lfloor l \cdot r_{i,j}\rfloor - 2, \ i.e., \\ cvr\text{-}dev_{i,j} &\leq 2. \end{aligned}$$

(2) The spacing smoothness bounds can be derived directly by Theorem 2.2. □

J. Proof of Theorem 4.5

Just consider the critical situation, which produces the largest delay for the cell at the head of VOQ. Let $Pos_{i,j}(k)$ denote the position of the kth cell ($k \geq 1$) of flow $f_{i,j}$ in the output sequence. Consider the interval $[I_{i,j}, I_{i,j} + \lceil (k+1)/r_{i,j}\rceil)$. By case (1) of Theorem 3.2, this interval covers output operations of at least $\lfloor \lceil (k+1)/r_{i,j}\rceil \cdot r_{i,j}\rfloor \geq k+1$. Since at most one ineffective output operation, this interval will cover at least k effective output operations. Therefore,

$$Pos_{i,j}(k) \leq I_{i,j} + \lceil (k+1)/r_{i,j}\rceil - 1.$$

We already know that the $(k-1)$th cell enters $XPB_{i,j}$ no earlier than its eligible time $I_{i,j} + \lceil (k-2)/r_{i,j}\rceil$, thus the kth cell becomes the head of $VOQ_{i,j}$ no earlier than $I_{i,j} + \lceil (k-2)/r_{i,j}\rceil + 1$. Therefore, from the kth cell becoming the head of $VOQ_{i,j}$ to its total departure from $XPB_{i,j}$, the delay is no more than

$$\begin{aligned} (I_{i,j} + \lceil (k+1)/r_{i,j}\rceil - 1) - (I_{i,j} + \lceil (k-2)/r_{i,j}\rceil + 1) \\ +1 \leq \lceil 3/r_{i,j}\rceil - 1 \leq \lfloor 3/r_{i,j}\rfloor. \end{aligned}$$

□

References

[1] F. Abel, C. Minkenberg, R. P. Luijten, M. Gusat, and I. Iliadis, "A four-terabit packet switch supporting long round-trip times," *IEEE Micro*, pp. 10–24, Jan./Feb. 2003.

[2] M. Ajmone-Marsan, P. Giaccone, E. Leonardi, and F. Neri, "Local scheduling policies in networks of packet switches with input queues," in *Proc. IEEE INFOCOM*, 2003.

[3] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, Nov. 1993.

[4] H. Balakrishnan, S. Devadas, D. Ehlert, and Arvind, "Rate guarantees and overload protection in input-queued switches," in *Proc. IEEE INFOCOM*, Mar. 2004.

[5] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches," in *Proc. IEEE INFOCOM*, 2000.

[6] C.-S. Chang, D.-S. Lee, and Y.-S. Jou, "Load balanced Birkhoff-von Neumann switches," *Comput. Commun.*, vol. 25, pp. 611–634, 2002.

[7] H. J. Chao, "Saturn: A terabit packet switch using dual round-robin," *IEEE Commun. Mag.*, vol. 38, no. 12, pp. 78–84, 2000.

[8] F. M. Chiussi and A. Francini, "Scalable electronic packet switches," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 4, pp. 486–500, May 2003.

[9] N. Chrysos and M. Katevenis, "Scheduling in non-blocking buffered three-stage switching fabrics," in *Proc. IEEE INFOCOM*, 2006.

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

[11] S.-T. Chuang, S. Iyer, and N. McKeown, "Practical algorithms for performance guarantees in buffered crossbars," in *Proc. IEEE INFOCOM*, 2005.

[12] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *Proc. IEEE INFOCOM*, 2000, pp. 556–564.

[13] Q. Duan and J. N. Daigle, "Resource allocation for quality of service provision in buffered crossbar switches," in *Proc. 11th ICCCN*, Oct. 2002, pp. 509–513.

[14] P. Giaccone, E. Leonardi, and D. Shah, "On the maximal throughput of networks with finite buffers and its application to buffered crossbar," in *Proc. IEEE INFOCOM*, 2005.

[15] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. 27, no. 10, pp. 1449–1455, Oct. 1979.

[16] K. G. I. Harteros, "Fast parallel comparison circuits for scheduling," Inst. Comput. Sci., FORTH, Heraklion, Crete, Greece, Tech. Rep. FORTH-ICS/TR-304.

[17] M. Hosaagrahara and H. Sethu, "Max-min fair scheduling in input-queued switches," *IEEE Trans. Parallel Distrib. Syst.*, in press. DOI: 10.1109/TPDS.2007.70746.

[18] J. Y. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*. Boston, MA: Kluwer Academic, 1990.

[19] T. Javidi, R. B. Magil, and T. Hrabik, "A high-throughput scheduling algorithm for a buffered crossbar switch fabric," in *Proc. IEEE ICC*, Jun. 2001, pp. 1586–1591.

[20] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos, "Variable packet size buffered crossbar (CICQ) switches," in *Proc. IEEE ICC*, Jun. 2004.

[21] I. Keslassy, M. Kodialam, T. V. Lakshman, and D. Siliadis, "On guaranteed smooth scheduling for input-queued switches," in *Proc. IEEE INFOCOM*, 2003.

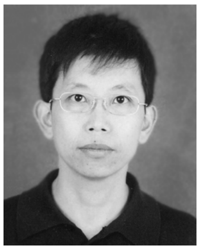
[22] P. Krishna, N. S. Patel, A. Charny, and R. J. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 6, pp. 1057–1066, Jun. 1999.

[23] H. T. Kung and R. Morris, "Credit-based flow control for ATM networks," *IEEE Network*, pp. 40–48, Mar./Apr. 1995.

[24] E. Leonardi, M. Mellia, F. Neri, and M. Ajmone-Marsan, "Bounds on delays and queue lengths in input-queued cell switches," *J. ACM*, vol. 50, no. 4, pp. 520–550, Jul. 2003.
[25] Y. Li, S. Panwar, and H. J. Chao, "On the performance of a dual round-robin switch," in *Proc. IEEE INFOCOM*, 2001, pp. 1688–1697.
[26] R. B. Magil, C. E. Rohrs, and R. L. Stevenson, "Output-queued switch emulation by fabrics with limited memory," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 4, pp. 606–615, May 2003.
[27] N. W. McKeown, "Scheduling algorithms for input-queued cell switches," Doctoral dissertation, Dept. EECS, Univ. of California, Berkeley, 1995.
[28] N. W. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1260–1267, Aug. 1999.
[29] L. Mhamdi and M. Hamdi, "CBF: A high-performance scheduling algorithm for buffered crossbar switches," in *Proc. HPSR*, Jun. 2003, pp. 67–72.
[30] C. Minkenberg, "Work-conservingness of CIOQ packet switches with limited output buffers," *IEEE Commun. Lett.*, vol. 6, no. 10, pp. 452–454, Oct. 2002.
[31] C. Minkenberg, R. P. Luijten, F. Abel, W. Denzel, and M. Gusat, "Current issues in packet switch design," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 119–124, 2003.
[32] C. Minkenberg, "Performance of i-SLIP scheduling with large round-trip latency," in *Proc. HPSR*, 2003, pp. 49–54.
[33] M. Nabeshima, "Performance evaluation of a combined input- and crosspoint-queued switch," *IEICE Trans. Commun.*, vol. E83-B, no. 3, pp. 737–741, Mar. 2000.
[34] D. Pan and Y. Yang, "Localized asynchronous packet scheduling for buffered crossbar switches," in *Proc. 2006 ACM/IEEE ANCS*, pp. 153–162.
[35] P. Pappu, J. Turner, and K. Wong, "Work-conserving distributed schedulers for terabit routers," in *Proc. ACM SIGCOMM*, 2004.
[36] T. Rodeheffer and J. B. Saxe, "Smooth scheduling in a cell-based switching network," Digital Systems Research Ctr., Palo Alto, CA, SRC Research Report 150, Feb. 1998.
[37] R. Rojas-Cessa, E. Oki, Z. Jing, and H. Chao, "CIXB-1: Combined input-one-cell-crosspoint buffered switch," in *Proc. IEEE HPSR*, 2001, pp. 324–329.
[38] R. Rojas-Cessa, E. Oki, and H. J. Chao, "On the combined input-crosspoint buffered switch with round-robin arbitration," *IEEE Trans. Commun.*, vol. 53, no. 11, pp. 1945–1951, Nov. 2005.
[39] D. N. Serpanos and P. I. Antoniadis, "FIRM: A class of distributed scheduling algorithms for high-speed atm switches with multiple input queues," in *Proc. IEEE INFOCOM*, 2000, pp. 548–555.
[40] D. C. Stephens and H. Zhang, "Implementing dsitributed packet fair queueing in a scalable switch architecture," in *Proc. IEEE INFOCOM*, 1998, pp. 282–290.
[41] J. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo*, Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Boston, MA: Kluwer Academic, 1997, p. 33.
[42] J. Turner, "Strong performance guarantees for asynchronous crossbar schedulers," in *Proc. IEEE INFOCOM*, 2006.
[43] K. Yoshigoe and K. J. Christensen, "An evolution to crossbar switches with virtual output queueing and buffered cross points," *IEEE Network*, pp. 48–56, Sep./Oct. 2003.
[44] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, no. 10, pp. 1374–1396, Oct. 1995.

Si-Min He (M'01) studied computer science at Tsinghua University, China, from 1986 to 1997 and received the Bachelor and Ph.D. degrees.

He is currently an Associate Professor at the Institute of Computing Technology, Chinese Academy of Sciences. His research is centered on the design, analysis, and application of high-performance algorithms in networking, multimedia streaming and bioinformatics.



Shu-Tao Sun received the Bachelor degree in information system engineering from the National University of Defense Technology, China, in 1990, and the Ph.D. degree from the Graduate University of Chinese Academy of Sciences, China, in 2006.

He is now an Assistant Professor at the Communication University of China. His research interests are in the areas of high-speed networking and multimedia streaming.

Hong-Tao Guan received the Bachelor degree in computer science and technology from Tsinghua University, China, in 2003, and since then he has been a Ph.D. student in the Department of Computer Science and Technology, Tsinghua University.

His research interests include scalable packet switch design.

Qiang Zheng received the Bachelor degree in computer science from NanKai University in 2004, and the Master degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2007.

His research interests include computer networks, performance evaluation, and algorithm design.

You-Jian Zhao received the Bachelor degree in computer science from Tsinghua University in 1991, and the M.S. and Ph.D. degrees in computer science from the Chinese Academy of Sciences in 1994 and 1999, respectively.

Now he is an Associate Professor at Tsinghua University. His current interests include computer network architecture, high speed routers, and large capacity switch fabric.

Wen Gao (M'92–SM'05) received the Ph.D. degree in electronics engineering from the University of Tokyo, Japan, in 1991.

He is a Professor at the Institute of Digital Media, Peking University, and an Adjunct Professor at the Chinese Academy of Sciences. His research interests include multimedia, video compression, face recognition, sign language recognition and synthesis, image retrieval, multimodal interface, and bioinformatics.