Reducing Crosspoint Buffers for Performance Guaranteed Asynchronous Crossbar Scheduling

Masoumeh Karimi, Zhuo Sun, and Deng Pan
Florida International University
Emails: {mkari001, zsun003, pand}@fiu.edu

***Abstract*—Buffered crossbar switches are special crossbar switches with an exclusive buffer at each crosspoint. They demonstrate unique advantages over traditional unbuffered crossbar switches, such as asynchronous scheduling and variable length packet handling. However, since crosspoint buffers are expensive on-chip memories, it is desired that each crosspoint has only a small buffer. In this paper, we propose a scheduling algorithm called Fair Asynchronous Segment Scheduling (FASS) for buffered crossbar switches, which reduces the crosspoint buffer size by dividing packets into shorter segments before transmission. FASS also provides tight constant performance guarantees by emulating the ideal Generalized Processor Sharing (GPS) model. Furthermore, FASS requires no speedup for the crossbar, lowering the hardware cost and improving the switch capacity. By theoretical analysis, we prove that FASS is strongly stable and therefore achieves 100% throughput. We also calculate the size bound for the crosspoint buffers and the reassembly buffers at output ports. Moreover, we show that FASS provides bounded delay guarantees. Finally, we present simulation data to verify the analytical results.**

I. Introduction

Buffered crossbar switches have recently attracted considerable attention as the next generation of high speed interconnects [1]. They are a special type of crossbar switches with an exclusive buffer at each crosspoint of the crossbar, which has been feasible with advances in modern VLSI technology to integrate miniaturized on-chip memories. Crosspoint buffers relax port contention, and greatly simplify the scheduling process. Buffered crossbar switches have thus demonstrated significant advantages over traditional unbuffered crossbar switches, such as asynchronous scheduling and variable length packet handling [2] [3] [4] [5].

However, crosspoint buffers are expensive on-chip memories, and the total crosspoint buffer size grows by the square of the switch size, i.e., N^2 crosspoint buffers for an $N \times N$ switch. For buffered crossbar switches to be practical, it is desired that each crosspoint has only a small size buffer, which is one of the main motivations of our work. In addition, it is obvious that the crosspoint buffer size depends on the maximum packet length. For packets in the Internet, although the maximum IP packet length is 1,500 bytes [6], about 60% of overall packets are less than 64 bytes, including TCP ACK and TCP control packets [7]. This indicates that even if we set large crosspoint buffers based on the maximum packet length, they cannot be efficiently utilized anyway. To address the issue, we propose in this paper a segment scheduling algorithm, which divides a packet into shorter segments before transmission. The maximum segment length can be arbitrarily small (in theory), leading to arbitrarily small crosspoint buffers. Note that our segmentation scheme is different from that for traditional unbuffered crossbar switches [8], because there are no padding bits for the last segment of a packet and thus no waste of bandwidth.

Another motivation of our work is to provide tight constant performance guarantees for buffered crossbar switches. The emulation of Push-In-First-Out (PIFO) Output Queued (OQ) switches is the main approach in the literature for crossbar switches to provide performance guarantees [9] [10] [11]. However, there are three main drawbacks with this approach. First, it has difficulty in providing tight performance guarantees, because it cannot emulate Worst-case Fair Weighted Fair Queueing (WF^{2}Q) [12], which is the only known fair queueing algorithm achieving constant performance guarantees [13]. Second, the emulation approach requires the switches to have speedup of at least two, which means that the crossbar needs to run twice faster than the input and output port. The speedup requirement increases the implementation cost and reduces the switch capacity. Third, the bandwidth allocation is not practical, because it does not consider bandwidth constraints at input ports, while flows may oversubscribe input ports [14]. In this work, we focus on addressing the first two drawbacks. Specifically, we use existing bandwidth allocation algorithms [15] [16] to calculate fair bandwidth allocation, and design a scheduling scheme to ensure the allocated bandwidth of each flow and achieve tight performance guarantees. In addition, our scheduling scheme requires no speedup for the crossbar.

In this paper, we propose a distributed scheduling scheme, called Fair Asynchronous Segment Scheduling (FASS), for buffered crossbar switches without speedup to achieve constant performance guarantees with reduced crosspoint buffers. First, we present a segmentation-and-reassembly (SAR) scheme to divide packets into short segments before transmission, so as to correspondingly reduce the crosspoint buffer size. We then propose the FASS algorithm to schedule segment transmissions, which uses a time stamp based approach to emulate the ideal Generalized Processor Sharing (GPS) [17] model and provides tight performance guarantees. By theoretical analysis, we prove that FASS is strongly stable and therefore achieves 100% throughput. We also calculate the size bound for the crosspoint buffers and the reassembly buffers at output ports. Moreover, we show that FASS provides bounded delay guarantees. Finally, we conduct simulations to verify the analytical results and measure the performance of FASS.

II. Fair Asynchronous Segment Scheduling (FASS)

In this section, we present the Fair Asynchronous Segment Scheduling (FASS) algorithm for buffered crossbar switches.

A. Packet Segmentation and Reassembly

As mentioned earlier, one of the motivations of our work is to make buffered crossbar switches practical by reducing crosspoint buffers. We present a packet segmentation-and-reassembly (SAR) scheme for this purpose. After a packet arrives at the input port, it will be divided into segments before transmission, if it is longer than a threshold, i.e. the maximum segment length. The maximum segment length can be arbitrarily small in theory, and a smaller maximum segment length leads to a smaller crosspoint buffer size. The segments will be used as the scheduling and transmission units. After they arrive at the output port, they will be reassembled back to the original packet before delivered to the output line.

B. Switch Architecture

The considered switch architecture includes N input ports and N output ports, connected by a buffered crossbar without internal speedup. Buffers are located at three possible bottlenecks: input ports, output ports, and crosspoints of the crossbar. Let In_i denote the i^{th} input port and Out_j denote the j^{th} output port. The available bandwidth of each input port and output port and also the crossbar is R. Each input port has a buffer to store arriving packets based on their destination output ports using Virtual Output Queues (VOQs) [4]. VOQs avoid the head of line (HOL) problem [6], which limits the maximum throughput of the switch [18]. Denote the virtual queue at In_i for packets destined to Out_j as Q_{ij}. Each crosspoint is equipped with an exclusive buffer represented by B_{ij} to connect In_i and Out_j. Each output port has a buffer to store received segments based on their source input ports using Virtual Input Queues (VIQs) [7]. VIQs are used to reassemble segments back into original packets before delivery to the output line.

C. Algorithm Description

FASS has two types of scheduling, called input scheduling and output scheduling. In input scheduling, an input port selects a segment from one of its N input queues and sends it to the corresponding crosspoint buffer. In output scheduling, an output port selects a segment from one of its N crosspoint buffers and retrieves it to the corresponding output queue.

We use the notation "I-O" to differentiate the algorithms for input scheduling and output scheduling, where "I" is the scheduler for input scheduling and "O" for output scheduling. "I" and "O" could be either FASS or GPS. If we do not care about the scheduler for output scheduling, we use a $*$ mark for "O". For example, FASS-GPS means that FASS is used for input scheduling and GPS for output scheduling. It is noted that, GPS is used as the ideal fairness model to compare the received service of a flow in our algorithm and in GPS.

Define the traffic from In_i to Out_j to be a flow F_{ij}. Use $r_{ij}(t)$ to represent the allocated bandwidth of F_{ij} at time t, which is calculated by specific bandwidth allocation algorithms [15] [16]. The calculated bandwidth should be feasible, i.e. no over subscription at any input or output port

$$\forall i, \sum_j r_{ij}(t) \le R, \text{ and } \forall j, \sum_i r_{ij}(t) \le R \tag{1}$$

To avoid input buffer overflow, input ports have admission control for each flow based on its allocated bandwidth. We use an extended leaky bucket for the admission control [6], which will be discussed in detail in Section III-A.

Input scheduling and output scheduling of FASS rely on only local information, and are conducted in an asynchronous and distributed manner. To be specific, an input port needs only the statuses of the queues in its input buffer, and does not exchange information with any crosspoint buffer or output port. Similarly, an output port needs only the statuses of its crosspoint buffers.

We first explain input scheduling. For easy presentation, let P_{ijk} represent the k^{th} arrived packet of F_{ij} and S^m_{ijk} represent the m^{th} segment of P_{ijk}. The first time stamp for S^m_{ijk} is called Virtual Input Start time VIS^m_{ijk}, which is the service start time of S^m_{ijk} at the input port in GPS-$*$. The second time stamp is Virtual Input Finish time VIF^m_{ijk}, i.e. the service finish time of S^m_{ijk} at the input port in GPS-$*$. In other words, if GPS is the input scheduler, VIS^m_{ijk} and VIF^m_{ijk} are the times that the first bit and the last bit of S^m_{ijk} leave Q_{ij}, respectively. VIS^m_{ijk} can be calculated as follows

$$VIS^m_{ijk} = \begin{cases} max\left(arv(in_{ijk}), VIF^{m'}_{ij(k-1)}\right), & m = 1 \\ VIF^{m-1}_{ijk}, & m \ge 2 \end{cases} \tag{2}$$

where $arv(in_{ijk})$ is the arrival time of P_{ijk} at the input port, and $VIF^{m'}_{ij(k-1)}$ is the virtual input finish time of the last segment $S^{m'}_{ij(k-1)}$ of the previous packet $P_{ij(k-1)}$. VIF^m_{ijk} satisfies the following relationship

$$\int_{VIS^m_{ijk}}^{VIF^m_{ijk}} r_{ij}(x)dx = l^m_{ijk} \tag{3}$$

where l^m_{ijk} is the length of S^m_{ijk}. Because $r_{ij}(x)$ has only discrete values in practice, VIF^m_{ijk} can be easily calculated. For example, if $r_{ij}(x)$ is a constant r_{ij} during [VIS^m_{ijk}, VIF^m_{ijk}], then VIF^m_{ijk} can be obtained as

$$VIF^m_{ijk} = VIS^m_{ijk} + (l^m_{ijk}/r_{ij}) \tag{4}$$

In the first step of input scheduling, In_i identifies eligible segments. A segment S^m_{ijk} is eligible for input scheduling if its virtual input start time VIS^m_{ijk} is smaller than or equal to the current system time t. In other words, a segment that has started transmission in GPS-$*$ is eligible in FASS-$*$. If there exist eligible segments in the input buffer, In_i will select among them the one S^m_{ijk} with the smallest virtual input finish time VIF^m_{ijk}, and send it to B_{ij}. If there are no eligible segments, In_i will wait until the next earliest virtual input start time of a segment. Note that when In_i is waiting for an eligible segment, if an empty input queue has a new segment, whose virtual input start time is equal to its arrival time, In_i should immediately start transmitting this new segment.

We denote the Actual Input Start time and Finish time of S^m_{ijk} in FASS-$*$ as AIS^m_{ijk} and AIF^m_{ijk}, which are the time that the first bit and the last bit of S^m_{ijk} leave Q_{ij} in FASS-$*$, respectively. Therefore

$$AIF^m_{ijk} = AIS^m_{ijk} + (l^m_{ijk}/R) \tag{5}$$

Output scheduling of FASS is similar to input scheduling. We define the Virtual Output Start time (VOS_{ijk}^m) and the Virtual Output Finish time (VOF_{ijk}^m) as the time that the first bit and the last bit of S_{ijk}^m leaves B_{ij} in FASS-GPS. In other words, after S_{ijk}^m is delivered to B_{ij} in FASS, if GPS is the output scheduler, S_{ijk}^m will start transmission at VOS_{ijk}^m and finish at VOF_{ijk}^m. Therefore, VOS_{ijk}^m is calculated as

$$VOS_{ijk}^m = \begin{cases} max\left(arv(crs_{ijk}^m), VOF_{ij(k-1)}^{m'}\right), & m = 1 \\ max\left(arv(crs_{ijk}^m), VOF_{ijk}^{m-1}\right), & m \geq 2 \end{cases} \quad (6)$$

where $arv(crs_{ijk}^m)$ is the arrival time of S_{ijk}^m at B_{ij} in FASS-$*$, equal to AIF_{ijk}^m, and $VOF_{ij(k-1)}^{m'}$ is the virtual input finish time of the last segment $S_{ij(k-1)}^{m'}$ of the previous packet $P_{ij(k-1)}$. Also, VOF_{ijk}^m satisfies the following relationship

$$\int_{VOS_{ijk}^m}^{VOF_{ijk}^m} r_{ij}(x)dx = l_{ijk}^m \quad (7)$$

Similarly, in the output scheduling of FASS, Out_j first identifies the eligible segments, and a segment S_{ijk}^m is eligible if its virtual output start time VOS_{ijk}^m is less than or equal to the current system time t. If there are eligible segments in the crosspoint buffers, Out_j selects the one S_{ijk}^m with the smallest virtual output finish time VOF_{ijk}^m among those eligible segments, and retrieves it to the output port. Otherwise, it waits for an eligible segment.

Correspondingly, AOS_{ijk}^m and AOF_{ijk}^m are the Actual Output Start time and Actual Output Finish time of S_{ijk}^m, which are the time that the first bit and the last bit of S_{ijk}^m leaves B_{ij} in FASS-FASS, respectively. It is obvious that

$$AOF_{ijk}^m = AOS_{ijk}^m + (l_{ijk}^m/R) \quad (8)$$

III. Performance Analysis

In this section, we theoretically analyze the performance of FASS. First, we introduce some properties and supporting lemmas. The proof is omitted due to space limitations.

Property 1: The actual input start time of any segment in FASS-$*$ is greater than or equal to its virtual input start time in GPS-$*$, i.e., $AIS_{ijk}^m \geq VIS_{ijk}^m$ (9)

Property 2: The actual output start time of any segment in FASS-FASS is greater than or equal to its virtual output start time in FASS-GPS, i.e.,

$$AOS_{ijk}^m \geq VOS_{ijk}^m \quad (10)$$

Lemma 1: The actual input start time of any segment in FASS-$*$ is less than or equal to its virtual input finish time in GPS-$*$, i.e., $AIS_{ijk}^m \leq VIF_{ijk}^m$ (11)

Now, let $toB_{ij}^{FASS}(t_1, t_2)$ and $toB_{ij}^{GPS}(t_1, t_2)$ represent the number of bits transmitted by F_{ij} from In_i to B_{ij} during time interval $[t_1, t_2]$ in FASS-$*$ and GPS-$*$, respectively.

Lemma 2: During the time interval [0,t], the difference between the number of bits sent from an input port In_i to a crosspoint buffer B_{ij} in FASS-$*$ and GPS-$*$ is l, where l is the maximum segment length, i.e.,

$$|toB_{ij}^{FASS}(0,t) - toB_{ij}^{GPS}(0,t)| \leq l \quad (12)$$

Lemma 3: The actual output start time of any segment in FASS-FASS is less than or equal to its virtual output finish time in FASS-GPS, i.e.,

$$AOS_{ijk}^m \leq VOF_{ijk}^m \quad (13)$$

Use $toO_{ij}^{FASS}(t_1, t_2)$ and $toO_{ij}^{GPS}(t_1, t_2)$ to represent the number of bits transmitted by F_{ij} from B_{ij} to Out_j during interval $[t_1, t_2]$ in FASS-FASS and FASS-GPS, accordingly.

Lemma 4: During the time interval [0,t], the difference between the number of bits sent from crosspoint buffer B_{ij} to output port Out_j in FASS-FASS and FASS-GPS is at most one maximum segment length, i.e.,

$$|toO_{ij}^{FASS}(0,t) - toO_{ij}^{GPS}(0,t)| \leq l \quad (14)$$

Lemma 5: During the time interval [0,t], the number of bits transmitted by flow F_{ij} from input port In_i to crosspoint buffer B_{ij} in GPS-$*$ is less than or equal to that from crosspoint buffer B_{ij} to output port Out_j in FASS-GPS plus $2l$, i.e.,

$$toB_{ij}^{GPS}(0,t) \leq toO_{ij}^{GPS}(0,t) + 2l \quad (15)$$

A. Switch Stability and Throughput

In this subsection, we prove that FASS achieves strong stability by showing that the length of input virtual queues are finite, which implies that FASS provides 100% throughput.

Let $X(t)$ be the vector of queue lengths at time t, and use $Q_{ij}(t)$ to show the occupancy of virtual queues such that $X(t) = (Q_{11}(t), Q_{12}(t), ..., Q_{ij}(t), ..., Q_{NN}(t))$. We follow the definitions in [22] and study the strong stability of our scheme, which implies 100% throughput [11].

Definition 1: $\|X(t)\|$ is called the Euclidean norm of vector $X(t)$, i.e., $\|X(t)\| = \sqrt{(Q_{11}(t), ..., Q_{ij}(t), ..., Q_{NN}(t))^2}$.

Definition 2: A system of queues is strongly stable if $\lim_{t\to\infty} sup\ E[\|X(t)\|] < \infty$.

The intuitive explanation is that segments belonging to packets of flow F_{ij} arrives and departs at the same rate, and they will not infinitely accumulate at either Q_{ij} or B_{ij} or O_{ij}.

As discussed in Section II-C, because a specific amount of bandwidth is allocated to each flow, it is necessary to have admission control for the flow to avoid input buffer overflow. The leaky bucket [6] is a widely used traffic shaping scheme, and we will use it for the admission control. In the classical definition of a leaky bucket, the flow rate is a constant, which we extend in this paper to be a variable. Use $toI_{ij}(t_1, t_2)$ to denote the number of incoming bits of F_{ij} during interval $[t_1, t_2]$. If F_{ij} is leaky bucket $(r_{ij}(t), \sigma_{ij})$ complaint, then during any interval $[t_1, t_2]$

$$toI_{ij}(t_1, t_2) \leq \int_{t_1}^{t_2} r_{ij}(x)dx + \sigma_{ij} \quad (16)$$

where σ_{ij} is called the burst size of F_{ij}. Intuitively, during any time interval, F_{ij} can have σ_{ij} more incoming traffic than what it can transmit.

Theorem 1: FASS is strongly stable when flows are leaky bucket compliant, i.e., FASS provides 100% throughput.

Proof: Assume that flow F_{ij} is leaky bucket $(r_{ij}(t), \sigma_{ij})$ complaint. Also assume that Q_{ij} is empty at s and $[s, t]$ is the last continuously backlogged period before t. This indicates that all segments belonging to packets of F_{ij} arriving at Q_{ij}

before s have finished transmission by s in GPS-$*$, and the next packet has not arrived yet. Thus

$$toI_{ij}(0,s) \leq toB_{ij}^{GPS}(0,s) + l \tag{17}$$

During $[s,t]$, Q_{ij} is continuously backlogged, and hence

$$toB_{ij}^{GPS}(s,t) = \int_s^t r_{ij}(x)dx \tag{18}$$

Because the incoming traffic is leaky bucket $(r_{ij}(t), \sigma_{ij})$ complaint, we have

$$toI_{ij}(0,s) \leq \int_s^t r_{ij}(x)dx + \sigma_{ij} \tag{19}$$

By (17), (18), and (19), we obtain

$$toI_{ij}(0,s) \leq toB_{ij}^{GPS}(0,t) + l + \sigma_{ij} \tag{20}$$

We know from Lemma 5 $toB_{ij}^{GPS}(0,t) \leq toO_{ij}^{GPS}(0,t) + 2l$, and from Lemma 4 that $toO_{ij}^{GPS}(0,t) \leq toO_{ij}^{FASS}(0,t) + l$, Hence,

$$toI_{ij}(0,t) - toO_{ij}^{FASS}(0,t) \leq 4l + \sigma_{ij} \tag{21}$$

By applying the fluid model [20] to our scheme and following the notations in [4], we obtain

$$Q_{ij}(t) + toB_{ij}^{FASS}(0,t) = toI_{ij}(0,t) - toO_{ij}^{FASS}(0,t) \tag{22}$$

Combining (21) and (22), we have

$$Q_{ij}(t) + toB_{ij}^{FASS}(0,t) \leq 4l + \sigma_{ij} \tag{23}$$

We know that l and σ_{ij} are bounded values, therefore

$$\lim_{t\to\infty}[Q_{ij}(t) + toB_{ij}^{FASS}(0,t)] \leq \lim_{t\to\infty}[4l + \sigma_{ij}] < \infty \tag{24}$$

Also we know that

$$Q_{ij}(t) \leq [Q_{ij}(t) + toB_{ij}^{FASS}(0,t)] \tag{25}$$

By comparing (24) and (25), we obtain $\lim_{t\to\infty} Q_{ij}(t) < \infty$ Accordingly, we have $\lim_{t\to\infty} Q_{ij}(t) < \infty$, and thus by using Definition 1, the Euclidean norm of vector $X(t)$, we obtain

$$\lim_{t\to\infty} supE[\|X(t)\|] < \infty \tag{26}$$

which satisfies Definition 2. Similar to [21], the occupancies of input queues serves to prove the stability of the scheduling algorithm, i.e., a scheduling policy that maintains stable queue length, provides the maximum achievable throughput. ■

B. Buffer Size Bound

To avoid overflow at crosspoint buffers and reassembly buffers, we would like to find the maximum number of bits buffered at any crosspoint and output port, respectively.

1) Crosspoint Buffer Size Bound: The following theorem gives the upper bound for the crosspoint buffer size.

Theorem 2: In FASS-FASS, the maximum number of bits buffered at any crosspoint buffer is upper bounded by four maximum segment length, i.e.,

$$toB_{ij}^{FASS}(0,t) - toO_{ij}^{FASS}(0,t) \leq 4l \tag{27}$$

Proof: By Lemma 4 $toO_{ij}^{FASS}(0,t) + l \geq toO_{ij}^{GPS}(0,t)$, and based on Lemma 5 $toO_{ij}^{GPS}(0,t) + 2l \geq toB_{ij}^{GPS}(0,t)$, and according to Lemma 2 $toB_{ij}^{GPS}(0,t) + l \geq toB_{ij}^{FASS}(0,t)$. Summing the above equations, we have proved the theorem. ■

2) Output Buffer Size Bound: Denote the maximum packet length as L and the maximum segment size as l. The following theorem gives the upper bound for the reassembly buffer size.

Theorem 3: In FASS-FASS, the maximum number of bits buffered in the reassembly buffers at the output port is upper bounded by $NL + l$.

Proof: As mentioned earlier in our switch model, there are N Virtual Input Queues (VIN) at each output port, corresponding to N input ports. New arrival segments are transferred through the crossbar and stored in output buffers as VINs. Thus, at most $N \times L$ bytes of memory space is needed per each output port for the reassembly process to recover the original packets. The worst case happens when an output port has received all segments belonging to packets from multiple input ports, excluding the last segment, and all of its VINs have been occupied in this way. If immediately upon arrival of the last segment of a packet, the first segment of another packet is consecutively retrieved by the same output port, more memory space will be required to buffer the new incoming segment and at the same time, forward the previously reassembled packet. Because segments of P_{ijk} need some time to finish the reassembly process before being sent out, the upper bound of the output buffer size is $NL + l$ to avoid overflow. It is noted that if SAR process is not applied, reassembly buffers will not be required and the output buffer size will be zero, as clearly shown in simulation results in Section IV-B2. ■

C. Delay Guarantees (Jitter)

In this subsection, we show that FASS-FASS can provide bounded delay guarantees. A packet can be departed when its last segment has been arrived to reassembly buffers at output ports, i.e., the departure time of a packet is equal to the departure time of its last segment. For easy analysis, we assume that the allocated bandwidth $r_{ij}(t)$ of F_{ij} is a constant r_{ij} during interval $[min(AIS_{ijk}^m, VIS_{ijk}^m), max(AOF_{ijk}^m, VOF_{ijk}^m)]$.

Use VOD_{ijk}^m to denote the Virtual Output Departure time of segment S_{ijk}^m in GPS-GPS. By neglecting the propagation delay, we have $VOD_{ijk}^m = VIF_{ijk}^m$. Similarly, AOD_{ijk}^m is the Actual Output Departure time of segment S_{ijk}^m in FASS-FASS, and $AOD_{ijk}^m = AOF_{ijk}^m$ if the propagation delay is neglected.

Theorem 4: The difference between the departure time of a packet in FASS-FASS and GPS-GPS is lower bounded to $-l_{ijk}^m(\frac{1}{r_{ij}} - \frac{2}{R})$ and upper bounded to $l(\frac{3}{r_{ij}} - \frac{2}{R})$, i.e.,

$$-l_{ijk}^m(\frac{1}{r_{ij}} - \frac{2}{R}) \leq AOD_{ijk}^m - VOD_{ijk}^m \leq l(\frac{3}{r_{ij}} - \frac{2}{R}) \tag{28}$$

Proof: We first prove the left side inequality as $-l_{ijk}^m(\frac{1}{r_{ij}} - \frac{2}{R}) \leq AOD_{ijk}^m - VOD_{ijk}^m$. It is obvious that

$$\begin{aligned} AOD_{ijk}^m &= AOF_{ijk}^m \\ &\geq arv(crs_{ijk}^m) + \frac{l_{ijk}^m}{R} = AIF_{ijk}^m + \frac{l_{ijk}^m}{R} \end{aligned} \tag{29}$$

Based on the Property 1, we know that $VIS_{ijk}^m \leq AIS_{ijk}^m$ or by (4) and (5), $VIF_{ijk}^m - \frac{l_{ijk}^m}{r_{ij}} \leq AIF_{ijk}^m - \frac{l_{ijk}^m}{R}$, and thus

$$\begin{aligned} VOD_{ijk}^m = VIF_{ijk}^m &\leq AIF_{ijk}^m + l_{ijk}^m(\frac{1}{r_{ij}} - \frac{1}{R}) \\ &\leq AOD_{ijk}^m + l_{ijk}^m(\frac{1}{r_{ij}} - \frac{2}{R}) \end{aligned} \tag{30}$$

Next, we prove $AOD^m_{ijk} - VOD^m_{ijk} \leq l(\frac{3}{r_{ij}} - \frac{2}{R})$. Based on Lemma 3, we know that $AOS^m_{ijk} \leq VOF^m_{ijk}$ and thus by (8), $AOF^m_{ijk} \leq VOF^m_{ijk} + \frac{l^m_{ijk}}{R}$. By Lemma 5, we know that $toB^{GPS}_{ij}(0,t) - toO^{GPS}_{ij}(0,t) \leq 2l$ and by Lemma 2, $toB^{FASS}_{ij}(0,t) - toB^{GPS}_{ij}(0,t) \leq l$. Combining them, we obtain $toB^{FASS}_{ij}(0,t) - toB^{GPS}_{ij}(0,t) \leq 3l$. This indicates that, after S^m_{ijk} arrives at B_{ij}, the maximum queue length at B_{ij} in FASS-GPS is $3l$. Because B_{ij} is served by GPS output scheduling with fixed allocated bandwidth r_{ij} in FASS-GPS, we have

$$VOF^m_{ijk} \leq arv(crs^m_{ijk}) + \frac{3l}{r_{ij}} \leq AIF^m_{ijk} + \frac{3l}{r_{ij}} \quad (31)$$

By Lemma 1 we know that, $AIS^m_{ijk} \leq VIF^m_{ijk}$ and thus $AIF^m_{ijk} \leq VIF^m_{ijk} + \frac{l^m_{ijk}}{R}$. Combining the above equations, we obtain

$$\begin{aligned} AOD^m_{ijk} = AOF^m_{ijk} &\leq VOF^m_{ijk} + \frac{l^m_{ijk}}{R} \\ &\leq AIF^m_{ijk} + \frac{3l}{r_{ij}} + \frac{l^m_{ijk}}{R} \leq VIF^m_{ijk} + \frac{3l}{r_{ij}} + \frac{2l^m_{ijk}}{R} \\ &\leq VOD^m_{ijk} + l(\frac{3}{r_{ij}} + \frac{2}{R}) \quad (32) \end{aligned}$$

■

IV. Simulation Results

We have performed simulations to evaluate the performance of FASS and verify the analytical results.

In our simulation, we consider a 16×16 buffered crossbar switch without speedup. Each input port and output port has a bandwidth of 1 Gbps. Since FASS is capable of handling variable length packets, we set the packet length in the range of [40,1500] bytes. We use the same model as in [19] for the bandwidth allocation. This model defines an allocated bandwidth $r_{ij}(t)$ of a flow F_{ij} at time t by applying an unbalanced probability w, i.e., $0 \leq w \leq 1$, as follows

$$r_{ij}(t) = \begin{cases} R\left(w + \frac{1-w}{N}\right), & \text{if } i = j \\ R\frac{1-w}{N}, & \text{if } i \neq j \end{cases} \quad (33)$$

To constrain the burstiness of a flow F_{ij}, we consider a leaky bucket $(\eta \times r_{ij}, \sigma_{ij})$, where η is the effective load and σ_{ij} is the burst size of F_{ij}. We set σ_{ij} of every flow to a fixed value of 10,000 bytes, and the burst may arrive at any time during a simulation run. We use two traffic patterns in the simulations. For the first pattern, each flow has a fixed allocated bandwidth during a single simulation run. The η is fixed to 1 and w takes one of the 11 possible values of [0,1] with a step of 0.1. For the second traffic pattern, a flow has a variable allocated bandwidth. The η takes one of the 10 possible values of [0.1,1] with a step of 0.1, and for a specific η value, a random permutation of the 11 different w values is used. We set the initial value of the crosspoint buffer size to 40 bytes and then adjust it from 100 to 1,500 bytes with a step of 100.

A. Throughput

To verify Theorem 1, we present the simulation data to show that our scheme achieves 100% throughput. Figure 1(a) illustrates the throughput under traffic pattern one with different segment sizes. We can observe that, the throughput for all unbalanced probabilities is greater than 99.99%, which demonstrates that FASS practically achieves 100% throughput. The throughput slightly decreases when the segment size increases, because with the same simulation time, larger segments sizes have smaller probabilities to finish the transmission of the last segment. Figure 1(b) depicts the throughput under traffic pattern two. As can be seen, the throughput grows consistently with the effective load, independent of the segment size, and finally reaches 100% when the effective load becomes 1. We can make the conclusion that, different segment sizes have no significant impact on the throughput performance.

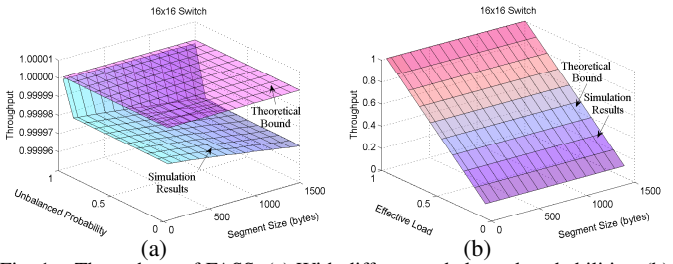


Fig. 1. Throughput of FASS. (a) With different unbalanced probabilities. (b) With different loads.

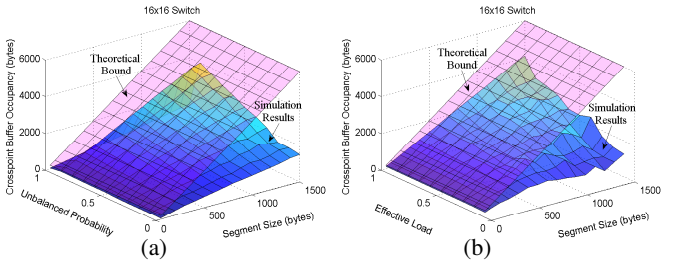
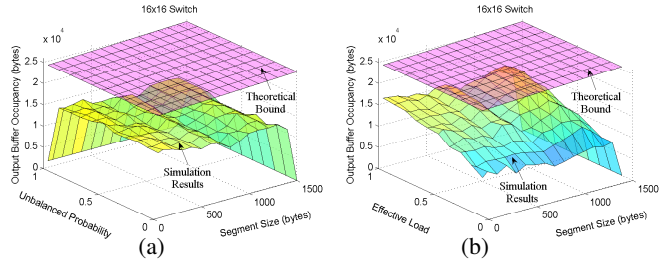


Fig. 2. Maximum Crosspoint buffer occupancy of FASS. (a) With different unbalanced probabilities. (b) With different loads.

B. Buffer Size Bounds

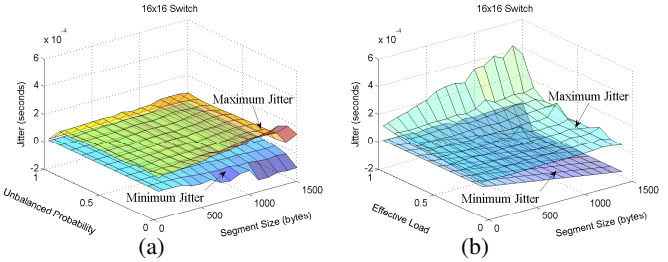
1) Crosspoint Buffer Size Bound: Theorem 2 gives the upper bound of the crosspoint buffer size as $4l$ bytes to avoid overflow. We now study the occupancy of crosspoint buffers via simulations. Figure 2(a) shows the maximum crosspoint buffer occupancy under traffic pattern one. It is observed that, the maximum occupancy is always smaller than the theoretical bound. The occupancy increases proportional to the segment size and grows as the unbalanced probability increases, but suddenly drops to about l bytes when the unbalanced probability becomes 1. Since at this point, all packets of In_i go to Out_i and hence, no switching is necessary. Figure 2(b) displays the maximum crosspoint buffer occupancy under traffic pattern two. We can see that, the maximum occupancy increases as the load and the segment size grow, but never exceeds the theoretical bound.

2) Output Buffer Size Bound: Theorem 3 gives the upper bound of the reassembly buffers at output ports as $NL + l$ bytes. Now, we evaluate the analytical results through the simulation data. Figure 3(a) demonstrates the maximum output buffer occupancy under traffic pattern one. As can be seen, the maximum occupancy is always less than the theoretical bound. As a special case, when the maximum segment size is set to the maximum packet length, i.e., 1,500 bytes, the output buffer occupancy suddenly drops to zero. In fact at this point, the incoming variable length packets are directly handled to the output lines without experiencing the SAR process. Note that, when the unbalanced probability becomes 1, the output buffer occupancy is significantly reduced to L, because all packets



(a) (b)

Fig. 3. Maximum Output buffer occupancy of FASS. (a) With different unbalanced probabilities. (b) With different loads.



(a) (b)

Fig. 4. Jitter of FASS. (a) With different unbalanced probabilities. (b) With different loads.

of In_i go to Out_i without switching. It means that, only one VIQ of each output port is occupied and the remaining $N-1$ VIQs are empty. Figure 3(b) plots the maximum output buffer occupancy under traffic pattern two. The maximum occupancy is always less than the theoretical bound and gradually grows when the effective load increases. Then, the occupancy suddenly drops to zero when the maximum segment size becomes 1,500 bytes, since the SAR process is no longer necessary.

C. Delay Guarantees (Jitter)

Finally, we present the simulation data on jitter, which is the difference between the departure time of a packet in FASS and GPS. Theorem 4 gives the lower bound and upper bound for the jitter as $-l_{ijk}^m(\frac{1}{r_{ij}}-\frac{2}{R})$ and $l(\frac{3}{r_{ij}}+\frac{2}{R})$, respectively.

Figure 4(a) shows the maximum and minimum jitter of a representative flow F_{11} under traffic pattern one. Since Theorem 4 assumes a fixed allocated bandwidth for r_{ij}, jitter depends on the segment size l_{ijk}^m. As can be seen, the minimum jitter is almost coincident with but always greater than or equal to the theoretical lower bound. The maximum jitter is always less than the theoretical upper bound and slightly increases as the segment size grows. However, the jitter suddenly drops when the unbalanced probability becomes one, as well as when the segment size reaches 1,500 bytes.

Figure 4(b) shows the maximum and minimum jitter of a representative flow F_{11} under traffic pattern two. As can be seen, the maximum jitter is always less than and close to the upper bound. It slightly increases as the segment size grows, and jumps when the effective load becomes one. The minimum jitter decreases when the segment size increases, but is always greater than the lower bound. Negative jitter means that most packets in FASS depart earlier than in GPS.

V. Conclusions

In this paper, we have proposed the Fair Asynchronous Segment Scheduling (FASS) algorithm for buffered crossbar switches. The main features of FASS can be summarized as follows. First, FASS reduces the crosspoint buffer size by dividing packets into segments before transmission. It needs no padding bits and thus does not waste the bandwidth. Second, FASS provides tight constant performance guarantees by tightly emulate the ideal GPS model. Third, FASS requires no speedup for the crossbar, reducing implementation cost and improving switch capacity. By theoretical analysis, we prove that FASS is strongly stable and therefore achieves 100% throughput. We also calculate the size bound for the crosspoint buffers and the reassembly buffers at output ports. Moreover, we show that FASS provides bounded delay guarantees. Finally, we have performed simulations, and the collected data demonstrate consistency with the analytical results.

References

[1] François Abel et al., "Design issues in next-generation merchant switch fabrics," *IEEE/ACM Trans. on Net.*, vol. 15 , no. 6, pp. 1603-1615, 2007.

[2] Yossi Kanizo, David Hay, and Isaac Keslassy, "The Crosspoint-Queued Switch," *IEEE INFOCOM '09*, Rio de Janeiro, Brazil, April 2009.

[3] M. Karimi, Z. Sun, D. Pan, and Z. Chen, "Packet-mode asynchronous scheduling algorithm for partially buffered crossbar switches," *IEEE Global Commun. Conf. (GLOBECOM 2009)* , Honolulu, HI, Nov. 2009.

[4] D. Pan and Y. Yang, "Localized Independent packet scheduling for buffered crossbar switches," *IEEE Trans. on Comp.*, vol. 58, no. 2, 2009.

[5] M. Katevenis et al, "Variable packet size buffered crossbar (CICQ) switches," *Proc. IEEE ICC 2004*, Paris, France, June 2004.

[6] J. Kurose and K. Ross, "Computer networking: a top-down approach," Addison Wesley, 4th edition, April 2007.

[7] W. Li and B. Liu, "SPF: to improve the performance of packet-mode scheduling," *ELSEVIER Comp. Com.*, vol. 28, pp. 1380-1391, Jul. 2005.

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

[9] B. Magill, C. Rohrs, and R. Stevenson, "Output-queued switch emulation by fabrics with limited memory," *IEEE Journal on Selected Areas in Communications*, vol 21, no. 4, pp. 606-615, May 2003.

[10] D. Pan, Z. Yang, K. Makki, and N. Pissinou, "Providing performance guarantees for buffered crossbar switches without speedup," *Int. Conf. on Heterogeneous Netw. for Quality, Reliability, Security and Robustness (QSHINE 2009)*, Las Palmas de Gran Canaria, Spain, Nov. 2009.

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

[12] S. Iyer and N. McKeown, "Analysis of the parallel packet switch architecture," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 2, pp. 314-324, Apr. 2003.

[13] J. Bennett and H. Zhang, "WF2Q: worst-case fair weighted fair queueing," *IEEE INFOCOM 1996*, San Francisco, CA, Mar. 1996.

[14] X. Zhang, S. Mohanty, L. Bhuyan, "Adaptive max-min fair scheduling in buffered crossbar switches without speedup," *INFOCOM*, AK, 2007.

[15] M. Hosaagrahara and H. Sethu, "Max-min fairness in input-queued switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 4, pp. 462-475, Apr. 2008.

[16] D. Pan and Y. Yang, "Max-min fair bandwidth allocation algorithms for packet switches," *IEEE IPDPS 2007*, Long Beach, CA, Mar. 2007.

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

[18] M. J. Karol, M. J. Hluchyj, and S. P. Morgan, "Input Versus Output Queueing on a Space-Division Packet Switch," *IEEE Transactions on Communications*, vol. 35, no. 12, pp. 1347-1356, Dec 1987.

[19] Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao, "CIXB-1: Combined input-once-cell-crosspoint buffered switch," *IEEE Workshop on High Performance Switching and RoutinG*, Dallas, TX, July 2001.

[20] J. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," *IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000.

[21] L. Mhamdi and M. Hamdi, "MCBF: a high-performance scheduling algorithm for buffered crossbar switches," *IEEE Communications Letters*, vol. 7, no. 9, pp. 451-453, Sep. 2003

[22] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet-mode scheduling in input-queued cell-based switches," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 666-678, 2002.