Parallel Packet Switch without Segmentation-and-Reassembly

Hao Jin, Deng Pan, and Niki Pissinou
Florida International University

***Abstract*—The ever increasing demand for more bandwidth at core routers has been a challenge for switch design. To address the challenge, parallel packet switches (PPSs) combine multiple parallel switching fabrics and provide huge aggregate bandwidth. However, most existing PPSs handle only fixed length packets, also called cells, mainly because traditional switching fabrics can process only cells. Since packets in the Internet are of variable length, existing PPSs need segmentation-and-reassembly (SAR) to process such packets, which will introduce padding bits and waste precious bandwidth. In this paper, we propose a PPS to directly handle variable-length packets without SAR. First, we present a simplified 1×1 variable-length PPS. We design the packet distribution and collection algorithms, and show that input and output conversion buffers are bounded by $2L$, where L is the maximum packet length. Next, we present a general $N \times N$ variable-length PPS, and propose the packet scheduling algorithms. We then prove our main result that such a PPS can emulate a first-in-first-out (FIFO) output queued (OQ) switch with speedup of two, i.e. emulating an FIFO OQ switch with bandwidth R by $2K - 1$ parallel switching fabrics each with bandwidth r, where $r = R/K$.**

I. Introduction

With the booming of broadband multimedia applications, there is an ever increasing demand for more bandwidth at core routers. However, traditional single switching fabric based switches are more and more difficult to meet this bandwidth demand both technically and financially. A popular solution to this challenge is the Parallel Packet Switch (PPS) [1], which combines several lower-speed switching fabrics or center stage switches (CSSs) to provide huge aggregate bandwidth.

Although there exist a number of PPS designs in the literature, most of them handle only fixed length packets, also called cells. [1] uses k output queued (OQ) switches as center stage switches, where k is the ratio between the external line rate and internal line rate. Each arriving packet is divided into fixed length cells first, and then sent into the switch. It is showed that, such a PPS can emulate a first-in-first-out (FIFO) OQ switch with speedup of 2, and emulate a push-in-first-out (PIFO) switch with speedup of 3. Further, [1] also shows that with extra buffer at each input and output, the PPS eliminates the need for speedup when emulating the FIFO OQ switch. [2] employs virtual input queues at multiplexers and corresponding cell dispatch-and-reassembly algorithms to achieve load balancing and in-order cell delivery. In [3], a large scale ATM switch similar to PPSs is presented. It uses a multipath parallel distribution approach to distribute ATM cells and thus provide large switching capacity. All the above PPSs are based on the assumption that packets are segmented into fixed length cells at the input and then reassembled back at the output. This segmentation-and-reassembly (SAR) [4] [5] process simplifies the switch design [6] [7], but may significantly affect the switch performance [8].

Recent advances in switching techniques have made it possible to directly process variable-length packets without SAR [8] [9]. Variable-length packet switches (or packet switches for short) have some unique advantages. First, packet switches can better utilize available bandwidth and achieve higher throughput. Cell switches may waste significant bandwidth on extra traffic including cell overheads and cell padding. In contrast, packet switches do not have such bandwidth waste. Second, since there is no segmentation and reassembly in packet switches, packets have shorter queuing delay than in cell switches. Therefore, packet switches reduce the latency that a packet experiences. Finally, no extra buffer space is needed at the input port or output port to segment and reassemble packets, which lowers hardware cost.

Our goal of this paper is to extend the packet switch design from single-fabric switches to PPSs, so that PPSs can utilize the advantages and achieve better performance with lower cost. Although there are a few studies on packet based PPSs, they are not able to process arbitrary variable-length packets. [10] handles variable-length packets by sending logical cells that belong to the same packet through the same switching plane. Although less overhead is needed, extra padding bits, which lower the overall throughput, are still required when the packet size is smaller than the cell size. [11] proposes the Flow-Mapping PPS (FM-PPS) with flow level load balancing. It guarantees the packet order of each micro-flow and thus eliminates the costly resequencing. However, FM-PPS works only when packets can be organized as micro flows and it needs k buffers at each demultiplexer which increases hardware cost.

In this paper, we present a PPS that directly handles variable-length packets without SAR and with low hardware cost. We start with a simplified 1×1 variable-length PPS, which is similar to the traditional inverse multiplexing system [12]. As shown in Figure 1, two additional types of buffers, namely input conversion buffers (ICBs) and output conversion buffers (OCBs), are required to accommodate the speed differences between the input/output lines and the CSSs. Two scheduling policies are designed to limit the size of ICBs and OCBs, respectively. We show that both the ICB size and OCB size can be bounded by $2L$, where L is the maximum packet length. Moreover, we prove that the second policy enables the switch to emulate an FIFO OQ switch. Next, we study

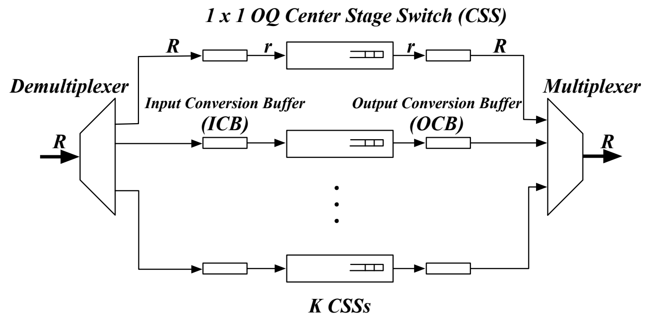
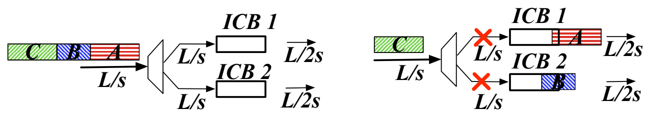


Fig. 1. 1×1 Variable-Length Parallel Packet Switch



(a) Time $0s$: packet A arrives (b) Time $5/3s$: packet C arrives

Fig. 2. A 1×1 vPPS with ICBs of size L is not work conserving.

the general $N \times N$ variable-length PPS by expanding the 1×1 switch structure and combing its two scheduling policies. We prove that such a PPS can emulate an FIFO OQ switch with speedup of 2, i.e. emulating an FIFO OQ switch with bandwidth R by using $2K-1$ CSSs each with bandwidth r, where $r = R/K$.

The rest of the paper is organized as follows. In Section II, we describe the 1×1 variable-length PPS. In Section III, we present the general variable-length PPS. Finally in Section IV, we conclude the paper.

II. 1×1 Variable-Length Parallel Packet Switch

Before presenting the general variable-length PPS (vPPS), we first describe a simplified vPPS with one input and one output, which will be the basis to design the scheduling algorithms for the general case.

A. Switch Structure

A 1×1 vPPS, as shown in Figure 1, consists of a demultiplexer with bandwidth R, K 1×1 CSSs each with bandwidth $r = R/K$, and a multiplexer with bandwidth R. The demultiplexer distributes variable-length packets to CSSs, from where the packets are collected by the multiplexer. Since a 1×1 switch needs no switching, the CSS in this case is simply a queue. The demultiplexer and the CSS have different bandwidth, and therefore each CSS needs an ICB to accommodate the speed difference. When the demultiplexer dispatches a packet to a CSS, the demultiplexer first sends the packet to the corresponding ICB, from where it will be retrieved by the CSS. If there are multiple packets in the ICB, they are stored as a first-in-first-out queue. Similarly, each CSS also has an OCB for the speed difference with the multiplexer. Note that the demultiplexer has no high-speed buffer to buffer arriving packets, and similarly the multiplexer has no high-speed buffer to store outgoing packets.

We are interested in the scheduling policies and conversion buffer sizes for the demultiplexer and multiplexer to be work conserving, i.e. keeping busy if there are packets to transmit. This seems trivial for fixed length cells, which can be easily accomplished with a round-robin packet distribution policy and L buffer space at each ICB and OCB, where L is the maximum packet length. However, with variable length packets, the problem becomes more challenging, which we will illustrate using the following example. Consider a 1×1 vPPS with two CSSs. The bandwidth of the demultiplexer and multiplexer is L/s, and that of the CSS is $L/2s$. Each ICB has L buffer space. Three packets A, B, and C with length of $L, 2L/3$, and L, respectively, arrive at the demultiplexer back to back, as shown in Figure 2(a). Without loss of generality, assume that at time $0s$ the demultiplexer start dispatching the first packet A to the first ICB, and the dispatch will finish at $1s$. Next, during $[1s, 5/3s]$, the demultiplexer dispatches the second packet B to the second ICB. Note that the first and second ICBs will not become empty earlier than $2s$ and $7/3s$, respectively. However, the demultiplexer finished dispatching packet B and should start dispatching the third one C at $5/3s$. Therefore, although the total bandwidth of the two CSSs is the same as that of the demultiplexer, but the demultiplexer cannot be work conserving with L ICB space.

In the rest of this section, we propose two scheduling policies for packet distribution and collection, and analyze the conversion buffer size bounds.

B. Policy A: ICB based Packet Distribution

This policy considers only packet distribution at the demultiplexer, and is based on the shortest queue first (SQF) algorithm. Specifically, when a new packet arrives, the demultiplexer checks the queue lengths of all the K ICBs, and selects the shortest queue to send the packet. If multiple ICBs have the same shortest length, the demultiplexer selects the one with the smallest index.

We will first analyze the characteristic of ICBs and prove later that, with Policy A, the size of any ICB is bounded by a small value, i.e. $2L$, while the demultiplexer is guaranteed to be work conserving. Denote the queue length of the ith ICB as $\widehat{B_i}$. (Since we always consider the values of different variables at the same time point, we omit the time parameter in the notation for simplicity.) Due to the space limitation, we only show the proofs of part of the lemmas and theorems.

Lemma 1: The queue length difference between any two ICBs is less than or equal to L, i.e.

$$|\widehat{B_i} - \widehat{B_j}| \leq L \tag{1}$$

Proof: Assume the queue length difference between any two ICBs could be greater than L, or $\widehat{B_i} - \widehat{B_j} > L$. Since the length $\widehat{L_n}$ of the last packet n of the ith ICBs is less than or equal to largest packet size L, we have $(\widehat{B_i} - \widehat{L_n}) - \widehat{B_j} > L - \widehat{L_n} > L - L = 0$. In other words, when packet n was selecting its CSS, it did not choose the one with the shortest ICB length which contradicts the policy. Hence the assumption is not possible. ■

Based on whether all the CSSs are retrieving packets from their ICBs, we define two statuses. In the *partially-busy* status, at least one CSS is idle (with empty ICBs), while in the *fully-busy* status, all CSSs are busy (without empty ICBs).

Theorem 1: In the partially-busy status, the queue length of any ICB is bounded by L, i.e.

$$\widehat{B_i} \leq L \tag{2}$$

Proof: In the partially-busy status, the minimum queue length of ICB equals to zero when the ICB is idle. Also from Lemma 1 we know that the maximum queue length difference between two queues are less than or equal to L. Thus the maximum ICB queue length is less than or equal to L when system is in the partially-busy status. ■

Lemma 2: In the fully-busy status, the total length of all ICBs is less than or equal to $(K-1)L$, i.e.

$$\sum_{i=1}^{K} \widehat{B_i} \leq (K-1)L \tag{3}$$

The proof is omitted due to space limitations.

Lemma 3: In the fully-busy status, the maximum value of the minimum ICB queue length is less than or equal to $(1-\frac{1}{K})L$, i.e.

$$\min\{\widehat{B_i}\} \leq (1-\frac{1}{K})L \tag{4}$$

Theorem 2: In the fully-busy status, the queue length of any ICB is bounded by $2L$, i.e.

$$\widehat{B_i} \leq 2L \tag{5}$$

Proof: Theorem 2 can be proved by Lemma 1 and 3. ■

Theorem 3: For a 1×1 vPPS adopting the SQF scheduling policy, the ICB queue length is bounded by $2L$, i.e.

$$\widehat{B_i} \leq 2L \tag{6}$$

Proof: By Theorem 1 and 2, Theorem 3 is proved. ■

C. Policy B: OCB based Packet Distribution and Retrieval

We now present the second policy for the 1×1 vPPS, which controls both the packet distribution/collection and the switching at CSS. First, we describe the detail process and parameter definitions of each phase, namely *Packet Distribution, Switching at CSSs* and *Packet Collection.* Then we show by lemmas and theorems that by employing Policy B, the 1×1 vPPS emulates FIFO OQ switch with small bounded OCB size.

1) Packet Distribution: When packet n arrives, the demultiplexer distributes packets to different CSSs. The basic idea is that the demultiplexer selects the CSS i with the earliest ***OCB Entry Start time*** $\widehat{OES}_{n,i}$, whose calculation will be explained in detail later. If multiple CSSs has the same earliest OCB entry start time, the one with the smallest index will be selected.

The calculation of $\widehat{OES}_{n,i}$, which represents packet n's entry start time of the ith OCB, is different for different packet categories. To simplify the analysis, we use a fixed time $\widehat{T}$ to represent the maximum delay from the input port to the CSS output queue. Denote the ***I**nput port **A**rrival time* of packet n as $\widehat{IA_n}$ and its ***C**SS output queue **A**rrival time* as $\widehat{CA_n}$. Then we have $\widehat{CA_n} = \widehat{IA_n} + \widehat{T}$.

For the first K packets, according to the policy, each of them selects the CSS with an empty OCB and with the same index. When these packets arrive at the CSS output queues, the OCB entry is available. However, they will not enter OCB but wait until time $\widehat{X}$, where $\widehat{X} = \widehat{CA_1} + \frac{L}{r}$. Thus, the first K packets have the same OCB entry start time, i.e.

$$\widehat{OES}_{1,1} = ... = \widehat{OES}_{K,K} = \widehat{X} = \widehat{CA_1} + \frac{L}{r} \tag{7}$$

On the other hand, when a packet after the first K one arrives at the CSS, the output queue may already have packets. Packet n will not start entering OCB until the last packet in the CSS output queue finishes its OCB entry. Denote the last packet in CSS i as packet m and its ***OCB Entry Finish time*** as $\widehat{OEF}_{m,i}$. Then $\widehat{OES}_{n,i}$ is calculated as

$$\widehat{OES}_{n,i} = max(\widehat{CA_n}, \widehat{OEF}_{m,i}); \forall n > K \tag{8}$$

Then the demultiplexer selects the CSS with the smallest OCB entry start time for packet n.

2) Switching at CSSs: As mentioned before, the CSS of 1×1 PPS can be treated as a queue. Thus, when packet n arrives at the CSS, it will stay in the output queue until the OCB entry start time comes and then start to enter the OCB.

3) Packet Collection: Finally, the multiplexer collects packets from OCBs. Specifically, the multiplexer collects packets one by one according to their arrival order to the input port. Recall that the first packet enters the OCB at time $\widehat{X}$. The multiplexer will start to collect the first packet at time $\widehat{D}$, where $\widehat{D} = \widehat{X} + \frac{L}{r}$. Denote the ***OCB Departure Start time*** of packet n from the ith OCB as $\widehat{ODS}_{n,i}$. Therefore,

$$\widehat{ODS}_{1,i} = \widehat{D} = \widehat{X} + \frac{L}{r} \tag{9}$$

Lemma 4: With Policy B, at any time after the OCB entry start time of the first packet, all OCB entries are busy.

Lemma 5: Packet n is already in the OCB when the multiplexer starts to collect it. In other words, packet n's OCB departure start time $\widehat{ODS}_{n,i}$ is greater than or equal to its OCB entry finish time $\widehat{OEF}_{n,i}$, i.e.

$$\widehat{ODS}_{n,i} \geq \widehat{OEF}_{n,i} \tag{10}$$

Proof: Since the multiplexer collects packets by their arriving order at rate R, we have

$$\widehat{ODS}_{n,i} = \widehat{ODS}_{1,i} + \frac{\sum_{x=1}^{n-1} L_x}{R} \tag{11}$$

By (9), we have

$$\widehat{ODS}_{n,i} = \widehat{X} + \frac{L}{r} + \frac{\sum_{x=1}^{n-1} L_x}{R} \tag{12}$$

While

$$\widehat{OEF}_{n,i} = \widehat{OES}_{n,i} + \frac{L_n}{r} \leq \widehat{OES}_{n,i} + \frac{L}{r} \tag{13}$$

The OCB entry start time $\widehat{OES}_{n,i}$ of packet n is maximized when all K CSSs has the same OCB entry start time. Since all OCBs start the packet entry at the same time and are always busy, the maximal $\widehat{OES}_{n,i}$ is calculated by

$$\widehat{OES}_{n,i} \leq \widehat{OES}_{1,1} + \frac{\sum_{x=1}^{n-1} L_x}{Kr} \tag{14}$$

By (7) and (14), (13) becomes

$$\widehat{OEF}_{n,i} \leq \widehat{X} + \frac{\sum_{x=1}^{n-1} L_x}{R} \tag{15}$$

The lemma is proved by subtracting (15) from (12). ■

Theorem 4: The 1×1 vPPS with Policy B emulates a 1×1 FIFO OQ switch.

Proof: The multiplexer collects packet one by one by their arriving order. In Lemma 5, we proved that every packet is ready in the OCB when the multiplexer starts to collect it. In other words, the multiplexer does not wait between two packet collections. Thus the switch is working conserving. On the other hand, the switch departure start time (OCB departure start time) of the first packet is bounded. Thus all packets leave the switch continuously with a bounded delay. Hence, the 1×1 vPPS with Policy B emulates an FIFO OQ switch. ■

Theorem 5: With Policy B, the queue length $\widehat{C_i}$ of any OCB is bounded by $2L$, i.e.

$$\widehat{C_i} \le 2L \tag{16}$$

Proof: The queue length of any OCB keeps increasing until the multiplexer collects packet from it. So when packet n is being collected by the multiplexer, the current queue length $\widehat{C_{n,i}}$ of the ith OCB equals to the differences between the total packets arrived $\widehat{E_{n,i}}$ and the total packets left $\widehat{D_{n,i}}$, i.e.

$$\widehat{C_{n,i}} = \widehat{E_{n,i}} - \widehat{D_{n,i}} \tag{17}$$

In Lemma 4, it is proved that all CSSs send packets to OCB continuously. Thus when packet n starts to depart from OCB i at time $\widehat{ODS_{n,i}}$, $\widehat{E_{n,i}}$ is calculated as

$$\widehat{E_{n,i}} = (\widehat{ODS_{n,i}} - \widehat{OES_{1,1}})r = \frac{\sum_{x=1}^{n-1} L_x}{K} + L \tag{18}$$

When the nth packet departs from the ith OCB, all the previous packets entered the same OCB has already left. Thus, $\widehat{D_{n,i}}$ should equal to the total length of all the previous packets in the same OCB, $\widehat{P_{n,i}}$. Since the OCB entry is always busy. $\widehat{P_{n,i}}$ is minimized when the OCB enter time of packet n is minimized. In this case, the output queue length of the ith CSS is L shorter than that of others. Then we have

$$\widehat{D_{n,i}} = \widehat{P_{n,i}} \ge \frac{\sum_{x=1}^{n-1} L_x - (K-1)L}{K} \tag{19}$$

Thus, by (18) and (19)

$$\widehat{C_{n,i}} = \widehat{E_{n,i}} - \widehat{D_{n,i}} = (2 - \frac{1}{K})L \le 2L \tag{20}$$

Hence, the queue length of any OCB i is bounded by $2L$. ■

III. General Variable-Length Parallel Packet Switch

In this section, we present the general vPPS. We first describe the switch architecture, scheduling algorithms and parameter definitions. Then, we show by analysis that vPPS can emulate an FIFO OQ switch with speedup of 2.

A. Switch Architecture

As shown in Figure 3, an $N \times N$ vPPS consists of N demultiplexers, $2K-1$ CSSs, and N multiplexers. The vPPS has bandwidth of R, and each CSS has bandwidth r, where $r = R/K$. Each demultiplexer acting as an input of the vPPS, distributes arriving packets to the CSSs. The packets are then transmitted through the CSSs, and finally collected by multiplexers, which act as outputs of vPPS. Similar with 1×1 vPPS, each input (output) of the CSS has an ICB (OCB) accommodate the bandwidth difference with the demultiplexer (multiplexer). Note that the demultiplexers and multiplexers do not have high speed buffers.

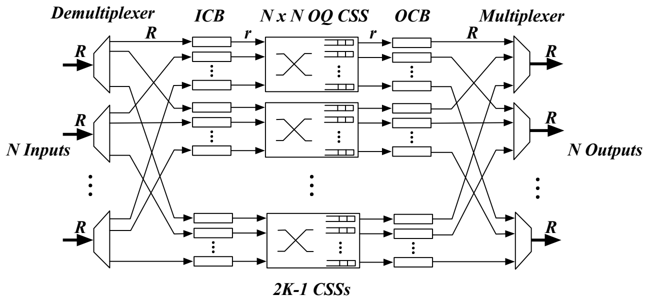


Fig. 3. General Variable Length Parallel Packet Switch

B. Scheduling Algorithms

The demultiplexers and multiplexers work as follows.

1) Packet Distribution: In general, the packet distribution in the vPPS can be divided into two stages, and each stage uses a policy similar to Policy A and Policy B of the 1×1 vPPS, respectively. In the first stage, the demultiplexer chooses K candidates out of all $2K-1$ CSSs based on their ICB length. Specifically, when packet n arrives, the demultiplexer checks the ICB status of each CSS and then selects the K CSSs with the shortest ICB queue length as candidates. In the second stage, the demultiplexer chooses the final CSS from the K candidates based on their OCB entry start time. To be specific, the demultiplexer selects the CSS i with the earliest ***O**CB **E**ntry **S**tart time* $OES_{n,i}$, whose calculation will be explained in detail later. If multiple CSSs has the same earliest OCB entry start time, the one with the smallest index will be selected.

The vPPS also calculates the $OES_{n,i}$ differently for different packet categories. Similarly, we use a fixed time T to represent the maximum delay from the input port to the CSS output queue. Denote the ***I**nput port **A**rrival time* of packet n as IA_n and its ***C**SS output queue **A**rrival time* as CA_n. Then we have $CA_n = IA_n + T$.

For the first K packets, each of them selects the CSS with an empty output queue and with the same index. For example, packet 1 will select CSS1. When these packets arrive at the CSS output queue, the OCB entry is available. However, they will not enter OCB until time X, where $X = CA_1 + \frac{L}{r}$. Thus, the first K packets have the same OCB entry start time, i.e.

$$OES_{1,1} = ... = OES_{K,K} = X = CA_1 + \frac{L}{r} \tag{21}$$

On the other hand, for packets after the first K ones, they may have to wait to enter the OCB until the last packet in the same CSS output queue finishes its OCB entry. Denote the last packet in CSS i as packet m and its ***O**CB **E**ntry **F**inish time* as $OEF_{m,i}$. The OCB entry start time of packet n is calculated as

$$OES_{n,i} = max(CA_n, OEF_{m,i}), \forall n > K \tag{22}$$

Then the demultiplexer selects the CSS with the smallest OCB entry start time for packet n.

2) Switching at CSSs: For simplicity, we assume all the CSSs are output queued switches. Therefore, after a packet arrives at the CSS, it will stay in the output queue until the OCB entry start time comes and then start to enter the OCB.

3) Packet Collection: The multiplexer collects packets one by one according to their arrival order at the input port. Recall that the first packet enters the OCB at time X. The multiplexer will start to collect the first packet at time D, where $D = X + \frac{L}{r}$. Denote the ***OCB Departure Start time*** of packet n from the ith OCB as $ODS_{n,i}$. Therefore,

$$ODS_{1,i} = D = X + \frac{L}{r} \tag{23}$$

C. Performance Analysis

Theorem 6: In the vPPS, the queue length B_i of any ICB is bounded by $2L$, i.e.

$$B_i \leq 2L \tag{24}$$

Proof: Recall that in the first stage of the packet distribution policy, all $2K-1$ CSSs will be divided in two groups, one with $K-1$ CSSs and another with K CSSs. Denote the former as Group 1 and the latter as Group 2. In the second stage, one of the CSS i in Group 2 will be chosen as the final CSS of the arrival packet. From the policy, we know that the ICB queue length of CSS i is less than or equal to that of any CSS in Group 1. Then if we consider CSS i and all $K-1$ CSSs in Group 1 together as a new Group 3, we find that the ICB queue length of CSS i is the shortest among the all K CSSs in Group 3. In other words, the CSS selection in vPPS can be considered as selecting the CSS with the shortest ICB length among K selected CSSs. Thus by Theorem 3, we can prove that the queue length of any CSS in Group 3 is bounded by $2L$. Since the CSS in Group 3 has the largest ICB queue length, the ICB queue length of CSSs in both Group 1 and Group 2 is bounded by $2L$. ■

Lemma 6: In the vPPS, packet n is already in the OCB when the multiplexer starts to collect it. In other words, packet n's OCB departure start time $ODS_{n,i}$ is greater than or equal to its OCB entry finish time $OEF_{n,i}$, i.e.

$$ODS_{n,i} \geq OEF_{n,i} \tag{25}$$

Proof: Lemma 6 can be proved by using the similar method in Lemma 5. ■

Theorem 7: The vPPS can emulate an FIFO OQ switch with speedup of 2.

Proof: Similar to the proof of Theorem 4, the FIFO OQ switch emulation can be proved by Lemma 6. The total bandwidth of the $2K-1$ CSSs is $(2-1/K)R$, which indicates speedup of 2. ■

Theorem 8: In the vPPS, the queue length C_i of any OCB i is bounded by $2L$, i.e.

$$C_i \leq 2L \tag{26}$$

Proof: The queue length of any OCB keeps increasing until the multiplexer collects packet from it. In other words, C_i may reach its maximum only at the time when packet n departs from the OCB, i.e. at $ODS_{n,i}$.

It is observed that, by time $ODS_{n,i}$, all the packets that entered OCB i prior to packet n have already left. Thus at this moment, the OCB length $C_{n,i}$ should equal to the total amount of packets that entered OCB i during $OES_{n,i}$ to $ODS_{n,i}$. And this total amount is maximized if the ith OCB entry keeps busy during $OES_{n,i}$ to $ODS_{n,i}$, i.e.

$$C_{n,i} \leq (ODS_{n,i} - OES_{n,i})r \tag{27}$$

By (21), (22) and (23), we have

$$C_{n,i} \leq (ODS_{n,i} - CA_n)r = 2L \tag{28}$$

Hence, the queue length of any OCB i is bounded by $2L$. ■

IV. Conclusion and Future Work

In this paper, we have studied the PPS architecture to directly handle variable-length packets without SAR. We first describe a 1×1 variable-length PPS, and present two scheduling policies, with which we prove that the input and output conversion buffers can be bounded by $2L$, where L is the maximum packet length. Next, we present the general $N \times N$ variable-length PPS. It extends the 1×1 PPS by expanding the switch structure and combining the two scheduling policies. It is showed that such a PPS can emulate an FIFO OQ switch, i.e. emulating an FIFO OQ switch with bandwidth R by $2K-1$ center stage switches each with bandwidth r, where $r = R/K$. Further we prove that input and output conversion buffers of size $2L$ are sufficient to achieve the emulation.

References

[1] S. Iyer, N. McKeown, "Analysis of the parallel packet switch architecture," *IEEE/ACM Trans. Networking*, vol. 11, no. 2, pp. 314-324, Apr. 2003.

[2] A. Aslam and K. J. Christensen, "A parallel packet switch with multiplexors containing virtual input queues," *Computer Communications*, vol. 27, no. 13, pp. 1248-1263, Aug. 2004.

[3] N. Moriwaki, A. Makimoto, Y. Oguri, M. Wada, and T. Kozaki, "Large scale ATM switch architecture for Tbit/s systems," *IEEE GLOBECOM*, Sydney, Australia, Nov. 1998.

[4] S. He et al., "On Guaranteed Smooth Switching for Buffered Crossbar Switches," *IEEE/ACM Trans. Networking*, vol. 16, no. 3, pp. 718-731, Jun. 2008.

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

[6] S. Mneimneh, V. Sharma, and K.-Y. Siu, "Switching using parallel input-output queued switches with no speedup," *IEEE/ACM Trans. Networking*, vol. 10, no. 5, pp. 653-665, Oct. 2002.

[7] B. Lin, I. Keslassy, "The concurrent matching switch architecture," *IEEE/ACM Trans. Networking*, vol.18, no.4, pp. 1330-1343, Aug. 2010.

[8] J. Turner, "Strong performance guarantees for asynchronous crossbar schedulers," *IEEE/ACM Trans. Networking*, vol. 17, no. 4, pp. 1017-1028, Aug. 2009.

[9] D. Pan and Y. Yang, "Localized independent packet scheduling for buffered crossbar switches," *IEEE Transactions on Computers*, vol. 58, no. 2, pp. 260-274, Feb. 2009.

[10] H. Zhong, D. Xu, Z. Zhu, "A parallel packet switch supporting variable-length packets," *IEEE ICCCAS*, Hong Kong, China, May 2005.

[11] L. Shi, G. Xia, and B. Liu, "Performance guarantees for flow-mapping parallel packet pwitch," *IEEE IPCCC*, New Orleans, LA, Apr. 2007.

[12] P. Fredette, "The past, present, and future of inverse multiplexing," *IEEE Communication Magazine*, vol. 32, pp. 42-46, Apr. 1994.