# The Concurrent Matching Switch Architecture

Bill Lin* Isaac Keslassy**

*University of California, San Diego, La Jolla, CA 92093–0407. Email: billlin@ece.ucsd.edu
**Technion – Israel Institute of Technology, Haifa 32000, Israel. Email: isaac@ee.technion.ac.il

***Abstract*— Network operators need high capacity router architectures that can offer scalability, provide throughput guarantees, and maintain packet ordering. However, current centralized crossbar-based architectures cannot scale to fast line rates and high port counts. On the other hand, while load-balanced switch architectures that rely on two identical stages of fixed configuration meshes appear to be an effective way to scale Internet routers to very high capacities, they incur a large worst-case packet reordering that is at best quadratic to the switch size. In this paper, we introduce the concurrent matching switch (CMS) architecture, which also uses two identical stages of fixed configuration meshes with the same scalability properties as current load-balanced routers. However, by adopting a novel contention-resolution architecture that is scalable and distributed, the CMS architecture enforces packet ordering throughout the switch. Using the CMS architecture, we show that scalability, 100% throughput, packet ordering, and $O(1)$ amortized time complexity with sequential hardware per linecard can all be achieved.**

## I. Introduction

### A. Background

Network operators need high capacity router architectures that can offer scalability, provide throughput guarantees, and maintain packet ordering. However, current crossbar-based router architectures with centralized scheduling and arbitrary per-packet dynamic switch configurations cannot scale to fast line rates and high port counts. Recently, there has been considerable interest in a class of switch architectures called *load-balanced routers* [1]–[5]. This class of architectures rely on two identical stages of fixed configuration meshes for routing packets. Figure 1 shows a diagram of a generic two-stage load-balanced switch architecture. The first mesh connects the first stage of input linecards to the center stage of intermediate input linecards, and the second mesh connects the center stage of intermediate input linecards to the final stage of output linecards. As shown in [3], this class of architectures appears to be a practical way to scale Internet routers to very high capacities and line rates. The scalability of this class of architectures can be attributed to two key aspects. First, they do not require a centralized scheduler: all queueing and decision-making functions can be performed locally at each linecard in $O(1)$ time. Second, these architectures are built using two identical stages of fixed configuration meshes whose deterministic interconnection patterns are independent of packet arrival. Thus, there is no need for arbitrary per-packet dynamic switch configurations, which can be extremely difficult to achieve at high-speeds. The use of fixed configuration meshes are particularly amenable to scalable implementations with optics, as exemplified by the 100 Tb/s reference design described in [3]. This reference design is based on a fixed hierarchical mesh of optical channels that interconnects $N = 640$ linecards, each operating at a rate of $R = 160$ Gb/s.

Although the load-balanced router architecture originally proposed in [1] is capable of achieving throughput guarantees, it has the critical problem that packet departures can be badly mis-sequenced. This is detrimental to Internet traffic since the widely used TCP transport protocol falsely regards out-of-order packets as indications of congestion and packet loss. Subsequently, several approaches have been proposed to address the packet ordering problem [2]–[4]. These approaches are based on bounding the amount of packet reordering through the switch and then using a finite reordering buffer at the output to correct mis-sequenced packets. However, these methods require reordering buffers of size $O(N^2)$, and the corresponding quadratic increase in packet delays, where $N$ is the switch size. Therefore, these approaches appear to be problematic, especially for the large switch sizes that load-balanced switch architectures target. In [5], another switch architecture based on the load-balanced switch architecture was proposed to address the packet ordering problem. However, this approach does not provide 100% throughput guarantees. Finally, recently, another switch architecture called the interleaved matching switch architecture was proposed in [6] that guarantees packet ordering. This architecture is also based on two fixed configuration meshes, exactly like the other load-balanced switch architectures. However, in contrast to the architecture presented in this paper, the interleaved matching switch architecture is based on a centralized scheduler model. In the case where the traffic profile is known *a priori* and remains stationary, it was shown in [6] that the centralized scheduling problem can be performed offline using matrix decomposition [7], and the online scheduling problem can be implemented in a distributed manner to provide service guarantees. Unfortunately, when the traffic is unpredictable and dynamically changing, the centralized scheduler requirement is not practical for large switch sizes.

In this paper, we introduce the concurrent matching switch (CMS) architecture. It is also based on two identical stages of fixed configuration meshes, so it inherits the same scalability properties of existing load-balanced routers for switch fabric implementation and is equally amenable to scalable implementation in optics. However, instead of bounding the amount of packet reordering through the switch, the CMS architecture *enforces* packet ordering throughout the switch by

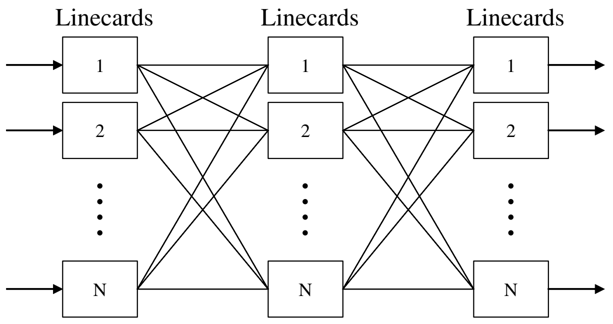

Fig. 1. Generic load-balanced switch architecture.

using a novel *scalable distributed load-balanced scheduling approach*. Instead of load-balancing *packets*, a CMS load-balances *request tokens* to each intermediate input linecard where each intermediate input linecard *concurrently* solves a *matching* problem based only on its local token count. By each intermediate input linecard solving a local matching problem in parallel, each intermediate input linecard independently selects a virtual output queue from each input linecard to service such that the packets selected can traverse the two fixed configuration meshes in parallel without conflicts. Packets from selected virtual output queues in turn depart in order from the input linecards, through the intermediate input linecards, and finally through the output linecards. Each intermediate input linecard has $N$ time slots to perform each matching, so the complexity of existing matching algorithms can be amortized by a factor of $N$. The exchange of tokens and packets occur over the two fixed uniform meshes without the need for arbitrary dynamic switch configurations, and all queueing and decision-making functions are performed locally at each linecard using only local state information.

### B. Contributions of the Paper

This paper makes the following major contributions. First, we prove that the CMS architecture can achieve 100% throughput under admissible arrival traffic by using any stable matching algorithm at each intermediate input linecard [8]–[14], including provably stable matching algorithms that do not require speedups [12]–[14].

Second, we demonstrate by means of empirical results that the CMS architecture can achieve low average packet delays using practically implementable matching algorithms without speedup. Our simulations show noticeably lower average packet delays compared with existing load-balanced switch architectures that maintain packet ordering.

Third, we show that the CMS architecture is indeed scalable by showing that a class of provably stable matching algorithms [13], [14] with good delay properties can be amortized to $O(1)$ complexity at each linecard using only sequential hardware and local state information at each linecard. The use of practical matching algorithms that only require $O(1)$ amortized complexity and sequential hardware means that both algorithmic and computational hardware complexities for each linecard are *independent* of $N$, which makes the architecture highly scalable when coupled with the scaling properties of uniform meshes in optics. As an example, we show that the use of a self-randomizing algorithm called SERENA [14] in a CMS can achieve provably 100% throughput, good average delays, and $O(1)$ amortized algorithmic complexity with sequential hardware, all without speedup.

Finally, we show that the idea of *load-balanced scheduling* used in the CMS architecture can also be used to improve the scalability of scheduling algorithms even in the case of single crossbar switch implementations.

### C. Organization of the Paper

The rest of the paper is organized as follows. In Section II, we introduce the CMS architecture. In Sections III and IV, we prove that the CMS architecture indeed achieves 100% throughput when used with a stable matching algorithm. In Section V, we briefly outline how the architecture can be optimized to improve delay bounds. In Section VI, we compare the average delay performance of the CMS architecture with existing load-balanced router architectures on both non-bursty and bursty traffic. In Section VII, we briefly outline how load-balanced scheduling can be used to improve the scalability of crossbar switches. Finally, we conclude the paper in Section VIII.

## II. The Switch Architecture

In this section, we provide a high-level overview of the CMS architecture. We defer to Sections III and IV for more detailed discussions on the operation of the CMS architecture. Note that throughout this paper, we assume that packets have a fixed length and time is slotted.

### A. Overview

The CMS architecture consists of three linecard stages that are interconnected by two fixed uniform meshes, exactly like the load-balanced switch architecture described in [1]–[4]. The CMS architecture is depicted in Figure 2. A high-level overview of the switch operation is as follows:

1) In the basic load-balanced switch architecture proposed in [1], incoming packets are *uniformly load-balanced* across the $N$ intermediate input linecards at the center stage where packets are buffered. Instead, in the CMS architecture, incoming packets are mainly buffered in *virtual output queues* at each input linecard. Specifically, each input linecard $i$ maintains $N$ virtual output queues, $Z_{i,1}, \ldots, Z_{i,N}$, one per output destination, as shown in Figure 2(a). Incoming packets to input $i$ destined for output $k$ are buffered at their virtual output queue $Z_{i,k}$ immediately upon arrival.
2) Instead of spreading packets across the center stage, a key idea in the CMS architecture is to first spread *request tokens* to the intermediate input linecards at the center stage instead of actual packets. Each request token acts as a *placeholder*. The actual packets are transferred later,

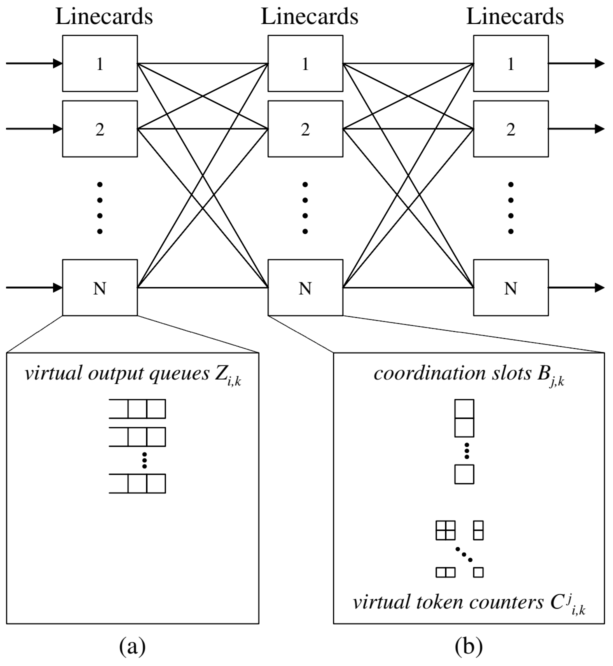
Linecards
Linecards
Linecards
1
2
N
1
2
N
1
2
N
virtual output queues $Z_{i,k}$
coordination slots $B_{j,k}$
virtual token counters $C^j_{i,k}$
(a)
(b)

Fig. 2. The Concurrent Matching Switch architecture. (a) Input linecard. (b) Intermediate input linecard.

based on *matching* decisions that are made by the intermediate input linecards. Specifically, each input linecard is periodically connected to a given intermediate input linecard every $N$ time-slots. For each incoming packet to input $i$ destined for output $k$, a request token $r(i, k)$ is immediately generated and sent to the intermediate input linecard that is currently connected to the input. In other words, the input linecard load-balances request tokens among intermediate linecards in a cyclical way that is influenced by the arrival time of each packet.

3) When a request token $r(i, k)$ is received at an intermediate input linecard $j$, the corresponding token counter at intermediate input $j$ gets incremented. Specifically, each intermediate input linecard $j$ maintains $N^2$ *virtual token counters*, $C^j_{i,1}, \dots, C^j_{i,N}$, one counter per each flow from input $i$ to output $k$, as shown in Figure 2(b). The virtual token counters are exactly analogous to the role of virtual output queues in conventional input-queueing (IQ) or combined input-output queueing (CIOQ) switches. However, instead of queueing actual packets, a virtual token counter $C^j_{i,k}$ keeps count of the number of request tokens that have been received at intermediate input $j$ for *flow* $(i, k)$.
4) Each intermediate input linecard then concurrently, and independently, solves a *matching* problem based on its own virtual token counts that it maintains locally. It does not need any global state information or any virtual token count information from any other intermediate input linecard. Any bipartite matching algorithm may be used with the CMS architecture to perform this matching step, leveraging the well-developed body of work in this area. As we shall see, each intermediate input linecard has $N$ time slots to perform each matching step, and thus the algorithmic complexity of the matching algorithm used may be *amortized* by a factor of $N$.
5) Based on the result of the matching step, each intermediate input $j$ sends in parallel over the first mesh a *grant token* $g(i, j, k)$ to each input $i$. The grant token indicates that the request token counter $C^j_{i,k}$ is positive and that the corresponding virtual output queue $Z_{i,k}$ has been matched. In other words, the grant token indicates that there was a demand and that the demand has been answered. In addition, token counters $C^j_{i,k}$ for which a grant token is generated are decremented.
6) In response to the grant token $g(i, j, k)$ received, each input $i$ then sends the packet at the head of the corresponding virtual output queue $Z_{i,k}$ to intermediate input $j$ over the first mesh. The (up to) $N$ packets sent by the inputs are then temporarily stored in a set of coordination slots at each intermediate input on their path to the outputs. Specifically, each intermediate input linecard $j$ maintains a set of $N$ coordination slots, $B_{j,1}, \dots, B_{j,N}$. As soon as the packets are fully received, the intermediate input linecard $j$ forwards them in parallel over the second mesh to the output linecards.
7) Finally, packets are received at output linecard $k$ where they depart immediately from the router.

As we shall see in Section III, using the above operation, the CMS architecture is *strongly stable* as long as a strongly stable matching algorithm is used for Step (4) above. In particular, for any admissible Bernoulli i.i.d. arrival traffic, CMS guarantees that the number of packets queued in the switch is not expected to grow to infinity. The proof relies on the interesting fact that the token traffic received by any intermediate input during $N$ time-slots has the same distribution as the packet traffic received by the router during a single time-slot.

However, it is well-known that a cyclical adversary arrival traffic can significantly reduce the throughput of the switch. Therefore, we will provide below a deterministic mechanism to fight the negative effects of cyclical adversary traffic patterns.

## *B. Adding Flow-Splitting*

To ensure that the CMS architecture is stable for any admissible traffic, we need to ensure that each input $i$ sends exactly $1/N^{th}$ of the request tokens generated for flow $(i, k)$ to each intermediate input $j$, thereby guaranteeing that exactly $1/N^{th}$ of the packets for flow $(i, k)$ will pass through intermediate input $j$.

To provide the guarantee that we evenly spread request tokens according to their flows, we extend the CMS architecture by adding a *flow splitter* and a set of load-balancing *request token queues* to each input linecard, as shown in Figure 3. Specifically, each input linecard $i$ maintains $N$ request token queues, $R_{i,1}, \dots, R_{i,N}$, one per intermediate input $j$, as shown in Figure 3(a). For each flow from input $i$ to output $k$, a pointer

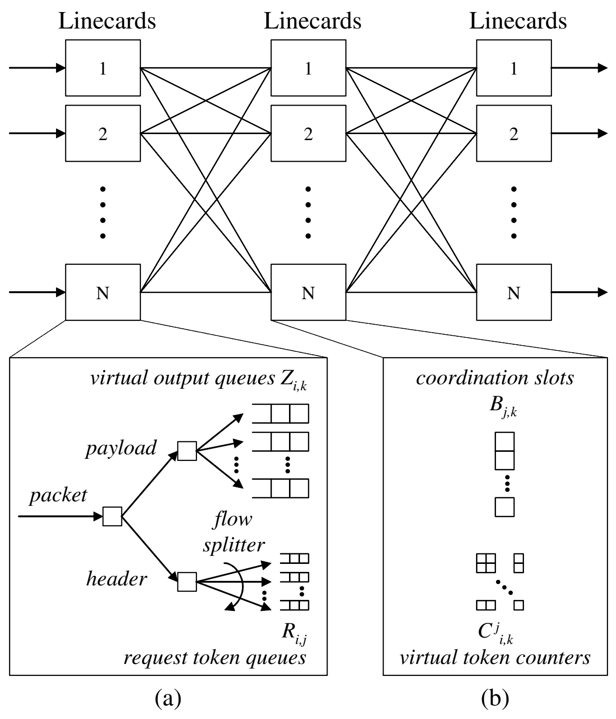

Fig. 3. The Concurrent Matching Switch architecture with *flow-splitting*. (a) Input linecard. (b) Intermediate input linecard.

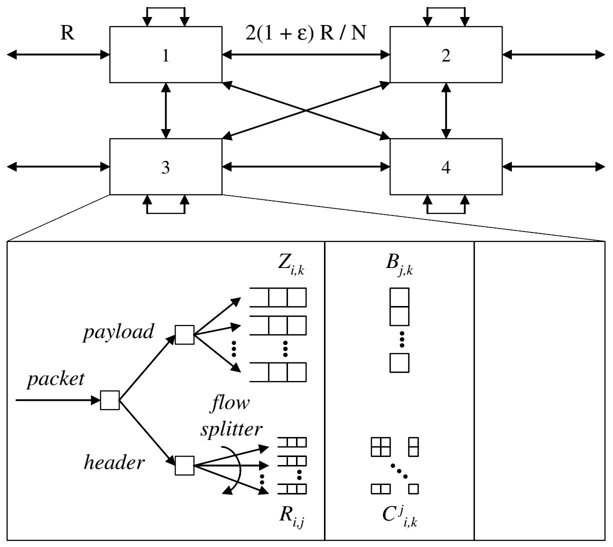

Fig. 4. The Concurrent Matching Switch architecture using a single combined mesh, shown with $N = 4$.

keeps track of the last request token queue in which a request token was placed, and the next request token is always placed in the next request token queue in round-robin order.

Using a flow-splitter, request tokens are distributed based on the order of the corresponding packet in its respective flow rather than based on the packet's arrival time. As we shall see in Section IV, the size of each request token queue $R_{i,j}$ is guaranteed to be less than $N$ with flow-splitting. The rest of the switch operation is as described above in Section II-A. As detailed in Section IV, when flow-splitting is used, the CMS architecture is *stable* (hence providing 100% throughput) for *any* admissible traffic satisfying a strong law of large numbers as long as a stable matching algorithm is used.

### C. Mesh Implementation

As noted in [3], the two uniform meshes used in the load-balanced switch architecture can be replaced by a single mesh running twice as fast. For the CMS architecture, the two uniform meshes used can also be replaced by a single mesh running twice as fast, with each linecard now containing three logical parts (input, intermediate input, and output). This is depicted in Figure 4. Figure 4 depicts the input linecard with flow-spitters and request token queues added, but the same approach of replacing two uniform meshes with a single combined mesh is equally applicable to the CMS architecture without flow-splitting. In the remainder of the paper, unless otherwise noted, we will still refer to the 3 logical parts as *input*, *intermediate input*, and *output*, respectively, and we will refer to the physical linecard that combines all three logical parts as the *combined linecard*.

Specifically, each pair of combined linecards in the CMS architecture are interconnected by four fixed rate channels: two control channels, and two data channels. The first control channel is used to transmit request tokens from input $i$ to intermediate input $j$. This *request* control channel only needs to operate at a rate of $\epsilon R/N$, where $\epsilon$ is the ratio of the token size to the fixed packet size. Since it is clear that a request token $r(i,k)$ received at intermediate input $j$ is from input $i$, the request token only needs to represent the *output* destination ID $k$. For a switch size of $N$, the request token can be represented in $\lceil \log N \rceil$ bits. Suppose the packet size is $L$. Then,

$$\epsilon = \frac{\lceil \log N \rceil}{L}. \tag{1}$$

For example, with $N = 1024$, $L = 512$ bits (64 bytes), then $\epsilon = 10/512 \approx 2\%$, so the size of the token is negligible relative to the packet size.

The second control channel is used to transmit grant tokens from intermediate input $j$ to input $i$. This *grant* control channel also only needs to operate at a rate of $\epsilon R/N$. This is because it is again clear that a grant token $g(i,j,k)$ received at input $i$ is from intermediate input $j$. Therefore, a grant token only needs to represent the output destination $k$ so that input $i$ knows to service its $k^{th}$ virtual output queue. Thus, the grant token can also be represented in $\lceil \log N \rceil$ bits.

Besides the two control channels, the two data channels are used to transfer packets. The *first-stage* data channel is used to transfer a packet from input $i$ to intermediate input $j$ over the first logical mesh, and the *second-stage* data channel is used to transfer a packet from intermediate input $j$ to output $k$ over the second logical mesh. Each data channel operates at a fixed rate of $R/N$. Combining all four channels together, each pair of combined linecards are interconnected with a combined bandwidth of $2(1 + \epsilon)R/N$. For example, with $N = 1024$ and $L = 512$ bits, the required combined bandwidth for connecting each pair of combined linecards is $2(1 + \epsilon)R/N \approx 2(1 + 0.02)R/N \approx 2R/N$, which is approximately the same as required for the load-balanced router described in [3].

As described in [3], [4], the uniform mesh model can be readily implemented at very high capacities and line rates using different types of switches, such as optical meshes with space and/or wavelength multiplexing, as well as time-multiplexed cyclic permutation switches (also called round-robin switches).

### D. Speedup

It is possible to generalize the CMS architecture by using some speedup $S$. In the CMS architecture using speedup, time-slots are replaced by *phases*, with $S$ phases per time-slot. Data channels of rate $R/N$ (respectively control channels of rate $\epsilon R/N$) are replaced by data channels of rate $SR/N$ (respectively control channels of rate $\epsilon SR/N$), and $S$ packets (respectively tokens) are sent at each time-slot whenever a single packet (respectively token) was sent without speedup. As we shall see in Section III-E, the CMS architecture with speedup $S$ is stable when using any matching algorithm that is stable with speedup $S$.

### E. Linecard Complexity

Each linecard only requires information available locally to perform all of its decision and queueing functions. The most complex part of the linecard implementation is the implementation of the matching step, which is logically performed at each intermediate input linecard. As mentioned, any bipartite matching algorithm may be used with the CMS architecture, and each intermediate input linecard has $N$ time slots to perform each matching step. Therefore, the algorithmic complexity of the matching algorithm used is *amortized* by a factor of $N$. For example, a self-randomizing matching algorithm called SERENA [14] with $O(N)$ algorithmic complexity has been shown to provide both stability and good delay properties. Amortized over $N$ time slots, the algorithmic complexity reduces to $O(1)$. Since SERENA only requires sequential hardware without speedup, the hardware complexity at each linecard can be made *independent* of $N$. All other control functions in the linecards also only require constant time operation, which makes the architecture both scalable and practical to implement.

In addition to control functions, memory is required for temporary storage. The bulk of the memory required is for implementing the virtual output queues at the input linecards to provide temporary buffering at times of congestion. Besides these congestion buffers, the memory required for the remaining storage functions is relatively modest. In particular, the number of coordinate slots and virtual token counters required at each intermediate input linecard is fixed with respect to $N$, independent of congestion. In the case where a flow-splitter is used, as depicted in Figure 3, memory is required to implement the request token queues. Since the tokens in these request token queues only need to encode output destinations, the size of these tokens is negligible relative to the size of the packets, as noted in Equation 1.

### F. Properties of the CMS Architecture

The CMS architecture has the following properties.

- *Packet ordering is maintained throughout the switch.* The CMS architecture enforces packet ordering throughout the switch by making sure that once a packet is *matched* for departure from the input stage, it arrives to its corresponding output linecard after a *fixed* propagation delay, where it then subsequently departs. The matching step performed by the intermediate input linecards guarantees that a packet can traverse the two stages of meshes without any contention once matched. Packets from selected virtual output queues in turn depart in order from the input linecards, through the intermediate linecards, and finally through the output linecards.
- *CMS is stable.* As detailed in Sections III and IV, CMS provides 100% throughput under diverse admissible traffic arrival patterns, both with and without flow-splitting.
- *CMS is practical to implement.* As discussed in Sections II-C and II-E, fixed configuration meshes can be scaled to very high speeds and port counts, and the amortized algorithmic complexity of all linecard operations can be made constant. The combination of these two factors makes the CMS architecture practical to implement and highly scalable.
- *Priorities are practical to implement.* It is straightforward to extend the CMS architecture to support $P$ priority levels with $P \times N$ virtual output queues at each input instead of $N$. The $P$ priority levels can for example be used to distinguish different service levels. Like a conventional IQ or CIOQ switch, when a flow from input $i$ to output $k$ is selected in a match, the virtual output queue from input $i$ to output $k$ with the highest priority level is serviced.

## III. Stability of the CMS Architecture

In this section we prove that the CMS architecture is stable when using any stable matching algorithm and apply this result to specific stable matching algorithms.

### A. CMS Architecture Model

Consider a CMS architecture with $N$ linecards. We will use standard notations and assumptions developed in the literature [4], [6], [8].

We will first consider the switch model without speedup. Time is slotted and packets arrive to the switch at the beginning of a time-slot. Each packet arrives at some time-slot $n$ to some input $i$ and is destined to some output $k$. Upon the arrival of the packet, a new request token is created for the intermediate input $j$ that is connected with input $i$ at time $n$. The request token is immediately sent to this intermediate input $j$. We will say that the token belongs to token flow $(i, j, k)$.

For the sake of simplicity, we will decompose the scheduling process into five consecutive phases taking $N$ time-slots each, and corresponding to the two control channels, the two data channels and the matching phase: (a) transmission of request tokens, (b) matching, (c) transmission of grant tokens, (d) transmission of packets through the first mesh and (e) transmission of packets through the second mesh. We refer to [6] for a discussion on the practicality of this model.

**(a)** At the start of time-slot $n$, after packet arrivals, each input linecard $i$ sends at most one request token to intermediate input linecard $j$, where

$$j = ((n-1) \mod N) + 1. \tag{2}$$

These request tokens of size $\epsilon$ are sent in parallel over their respective request control channels at rate $\epsilon R/N$. Consequently, they take $N$ time-slots to propagate, and intermediate input linecard $j$ receives up to $N$ request tokens in parallel by the end of time slot

$$n + N - 1. \tag{3}$$

**(b)** At the start of time-slot

$$n + N, \tag{4}$$

after reception of the request tokens, each intermediate input linecard $j$ applies its matching algorithm $m$ to determine a one-to-one match between the set of inputs $i$ and the set of outputs $k$. As explained later, we assume that this matching algorithm takes $N$ time-slots to run, and therefore is done by the end of time-slot

$$(n+N) + (N-1) = n + 2N - 1. \tag{5}$$

**(c)** At the start of time-slot

$$n + 2N, \tag{6}$$

after the matching algorithm is completed, each intermediate input linecard $j$ sends up to $N$ grant tokens of size $\epsilon$ in parallel to the $N$ input linecards over their respective grant control channels at rate $\epsilon R/N$. These grant tokens take $N$ time-slots to propagate, and each grant token $g(i,j,k)$ reaches input linecard $i$ by the end of time slot

$$(n+2N) + (N-1) = n + 3N - 1. \tag{7}$$

Note that grant tokens are only generated if there exists some corresponding request token, i.e. if the virtual token counter is positive. Consequently, each grant token generates a later departure of a packet.

**(d)** At the start of time-slot

$$n + 3N, \tag{8}$$

after reception of grant token $g(i,j,k)$ from intermediate input linecard $j$, input linecard $i$ sends the head-of-line packet of virtual output queue $Z_{i,k}$ to intermediate input linecard $j$. The intermediate input linecard is selected following the equation used by the control channel (Equation (2)). Note that

$$\begin{aligned} j &= (((n+3N)-1) \mod N) + 1 \\ &= ((n-1) \mod N) + 1, \end{aligned} \tag{9}$$

which corresponds indeed to the intermediate linecard $j$ that granted the token, as indicated in Equation (2). These packets are sent in parallel over the first mesh at rate $R/N$. Consequently, they take $N$ time-slots to propagate, and intermediate input linecard $j$ receives up to $N$ packets in parallel by the end of time slot

$$n + 4N - 1. \tag{10}$$

**(e)** Finally, at the start of time-slot

$$n + 4N, \tag{11}$$

intermediate input linecard $j$ sends up to $N$ packets in parallel over the second mesh to the $N$ output linecards, including at most one packet to each output linecard $k$ from the corresponding slot $B_{j,k}$. Each output linecard $k$ then receives the packet sent by intermediate input linecard $j$ by the end of time slot

$$(n+4N) + (N-1) = n + 5N - 1, \tag{12}$$

and the packet departs immediately from the router.

Incidentally, note that the delay between the time-slot at which input $i$ sends its packet and the time-slot at which the packet finishes to arrive at output $k$ is

$$(n + 5N - 1) - (n + 3N) = 2N - 1, \tag{13}$$

which is completely independent of time-slot $n$ and intermediate input $j$. As noted before, this ensures that there is no reordering, since *a packet sent first also arrives first*.

## B. Notations

Let $A_{ijk}(n)$ denote the cumulative number of request tokens created for token flow $(i,j,k)$ by time-slot $n$. In other words, $A_{ijk}(n)$ denotes the number of packets arrived by time-slot $n$ to input linecard $i$, destined to output linecard $k$, and for which the token request is destined to intermediate input linecard $j$. We adopt the convention that $A_{ijk}(0) = 0$ for all $i,j,k$.

Similarly, let $R_{ijk}(n)$ denote the cumulative number of request tokens for flow $(i,j,k)$ arrived to intermediate input $j$ by time-slot $n$, $G_{ijk}(n)$ denote the cumulative number of grant tokens generated for flow $(i,j,k)$ by time-slot $n$, and $D_{ijk}(n)$ denote the cumulative number of packets corresponding to flow $(i,j,k)$ and having departed the router by the end of time-slot $n$. As noted in the above model, in the CMS architecture, each request token generated at time $n$ arrives at the intermediate input at the end of time-slot $n+N-1$, and therefore

$$R_{ijk}(n + N - 1) = A_{ijk}(n). \tag{14}$$

Similarly, each grant token generated at time $n + 2N$ corresponds to a packet departure at the end of time-slot $n+5N-1$, and therefore

$$D_{ijk}(n + 5N - 1) = G_{ijk}(n + 2N). \tag{15}$$

We also want to define the number of request tokens and packets queued in the switch. Let

$$Q_{ijk}(n) = R_{ijk}(n) - G_{ijk}(n) \tag{16}$$

denote the cumulative number of request tokens for flow $(i, j, k)$ that are still queued (have not yet been granted) by time-slot $n$, and let

$$X_{ijk}(n) = A_{ijk}(n) - D_{ijk}(n) \tag{17}$$

denote the cumulative number of packets for flow $(i, j, k)$ that are still queued by time-slot $n$.

Finally, let $A_{ik}(n)$ denote the cumulative number of arrivals to input $i$ by time-slot $n$ of packets destined to output $k$. Then

$$A_{ik}(n) = \sum_{j=1}^{N} A_{ijk}(n). \tag{18}$$

Similarly, the cumulative number of packet departures from the router will be denoted as

$$D_{ik}(n) = \sum_{j=1}^{N} D_{ijk}(n). \tag{19}$$

*C. Definition of Strong Stability*

We will now prove the strong stability of the CMS architecture by relying on the Lyapunov method. For the sake of conciseness, we will not develop again the whole framework of this method – the interested reader can refer to the large literature about Lyapunov techniques and notations in switches [9], [12], [13], [15].

We assume that the packet arrival process to each input is Bernoulli i.i.d., and that each packet has a given probability of being destined to any given output $k$ provided by a traffic matrix $\Lambda = [\lambda_{ik}]$. Further, we assume that the arrival matrix is doubly sub-stochastic (admissible), i.e., for all $i, k$,

$$\sum_{i=1}^{N} \lambda_{ik} \leq 1, \sum_{k=1}^{N} \lambda_{ik} \leq 1. \tag{20}$$

We can now introduce the definition of strong stability.

*Definition 1 (Strong Stability):* A switch is said to be *strongly stable* if under the Bernoulli i.i.d. admissible packet arrival process defined above, the number of packets queued in the switch is not expected to grow to infinity, i.e.,

$$\limsup_{n\to\infty} E\left[\sum_{i,j,k} X_{ijk}(n)\right] < \infty. \tag{21}$$

*D. Strong Stability Theorem*

The following theorem establishes that when each intermediate input linecard uses a strongly stable matching algorithm, then the CMS architecture is strongly stable as well. Please refer to Appendix I for the proof.

*Theorem 1 (CMS Strong Stability):* CMS is strongly stable when using any strongly stable matching algorithm.

In particular, this theorem applies to the MWM (Maximum Weight Matching) scheduling algorithm, which is known to be strongly stable [12].

*Corollary 1 (CMS-MWM):* CMS is strongly stable when using MWM.

Similarly, this theorem applies to the SERENA scheduling algorithm, which is known to be strongly stable as well [14].

*Corollary 2 (CMS-SERENA):* CMS is strongly stable when using SERENA.

*E. Stability with Speedup*

As discussed in Section II-D, the CMS architecture can be generalized to operate under some speedup of $S$. For more information on how to implement speedup, refer to [6]. As the following theorem shows, strong stability of a matching algorithm extends to the CMS architecture when using speedup as well. The proof is in Appendix II.

*Theorem 2 (Speedup):* The CMS architecture with speedup $S$ is strongly stable when using any matching algorithm that is strongly stable with speedup $S$.

In particular, this theorem applies to *maximal* matching algorithms such as iSLIP [10], since they are known to be stable with speedup two [9].

*Corollary 3 (CMS-Maximal):* CMS is strongly stable when using any maximal matching algorithm and speedup two.

## IV. Stability of CMS with Flow-Splitting

We proved in the previous section that the CMS architecture is strongly stable when using a strongly stable matching algorithm. In this section, we prove that the CMS architecture with flow-splitting is stable as well when using stable matching algorithms. In particular, we will prove the stability of CMS with flow-splitting using fluid models (we call it *stability* to distinguish from the *strong stability* defined previously and based on Lyapunov models). We will then apply this result to specific strongly stable matching algorithms.

*A. Definition of Stability*

The CMS architecture model and notations with flow-splitting are the same as the CMS model and notations defined previously.

As in [8], we assume that the arrival processes satisfy a strong law of large numbers (SLLN): for all $i, k$, with probability one,

$$\lim_{n\to\infty} \frac{A_{ik}(n)}{n} = \lambda_{ik}, \tag{22}$$

where $\Lambda = [\lambda_{ik}]$ forms the arrival rate matrix. This occurs, for instance, if they are jointly stationary and ergodic with arrival rates $\lambda_{ik}$. Further, we assume that the arrival matrix is doubly sub-stochastic (admissible), i.e., for all $i, k$,

$$\sum_{i=1}^{N} \lambda_{ik} \leq 1, \sum_{k=1}^{N} \lambda_{ik} \leq 1. \tag{23}$$

*Definition 2 (Stability):* A switch is said to be *stable* if under any arrival process satisfying Equations (22) and (23), then for all $i, k$, with probability one,

$$\lim_{n\to\infty} \frac{D_{ik}(n)}{n} = \lambda_{ik}. \tag{24}$$

## B. *Stability Theorem*

The following theorem establishes that when each intermediate input linecard uses a stable matching algorithm, then the CMS architecture with flow-splitting is stable as well. It is proved in Appendix III.

*Theorem 3 (CMS Stability with Flow-Splitting ):* CMS with flow-splitting is stable when using any stable matching algorithm.

In particular, this theorem applies to the MWM (Maximum Weight Matching) scheduling algorithm [12]. MWM is known to be stable [8].

*Corollary 4 (CMS-MWM):* CMS with flow-splitting is stable when using MWM.

## C. *Speedup*

We saw how to generalize the CMS architecture by using some speedup $S$. We can do the same for CMS with flow-splitting. As the following theorem shows, stability of a matching algorithm extends to the CMS architecture with flow-splitting when using speedup as well. The proof is in Appendix IV.

*Theorem 4 (Speedup):* The CMS architecture with flow-splitting and speedup $S$ is stable when using any matching algorithm that is stable with speedup $S$.

In particular, this theorem applies to *maximal* matching algorithms such as iSLIP [10], since they are known to be stable with speedup two [8].

*Corollary 5 (CMS-Maximal):* CMS with flow-splitting is stable when using any maximal matching algorithm and speedup two.

## V. Delay Optimizations

In this section, we informally outline how propagation delays can be improved. In Section II-C, the two control channels and the two data channels are described as four *separate* fixed rate channels, two control channels at fixed rate $\epsilon R/N$, and two data channels at fixed rate $R/N$, respectively. Specifically, to ease the explanation of the CMS architecture model in Section III-A, the scheduling process was described in terms of 5 consecutive phases, with the transmission of request tokens (phase (a)) and the transmission of grant tokens (phase (c)) separated into their own distinct phases from the transmission of packets through the first and second mesh stages (phases (d) and (e)), respectively. The transmission of request tokens of size $\lceil \log N \rceil$ was assumed to take $N$ consecutive time-slots to complete over a fixed channel of rate $\epsilon R/N$. Similarly, the transmission of grant tokens of size $\lceil \log N \rceil$ was also assumed to take $N$ consecutive time-slots to complete. Therefore, as stated in Equation 12, it takes in the best case $5N - 1$ time-slots after packet arrival before a packet can depart from its output destination.

However, we can improve upon this delay by *time-multiplexing* each control channel of rate $\epsilon R/N$ with a corresponding data channel of rate $R/N$ on to a single channel of rate $(1+\epsilon)R/N$. Specifically, we can time-multiplex the *request* control channel and the *first-stage* data channel on to the same shared channel at rate $(1+\epsilon)R/N$, and we can time-multiplex the *grant* control channel and the *second-stage* data channel on to another shared channel at rate $(1+\epsilon)R/N$. This way, we can combine phases (a) and (d) into a single phase, and we can combine phases (c) and (e) into another single phase. In particular, transmitting over a fixed channel at rate $(1+\epsilon)R/N$, the request tokens from the input stage will arrive at the corresponding intermediate inputs at the center stage $\epsilon N/(1+\epsilon)$ time-slots after departure from the input stage. Then, the transmission of packets through the first logical mesh can take the remaining $N/(1+\epsilon)$ time-slots to complete the transmission. Similarly, transmitting over a fixed channel at rate $(1+\epsilon)R/N$, the grant tokens from the center stage to the input stage will arrive to the input stage $\epsilon N/(1+\epsilon)$ time-slots after departure from the center stage, and the remaining $N/(1+\epsilon)$ time-slots can be used to transmit packets through the second logic mesh from the center stage to the output stage. Therefore, the overall best-case propagation delay (including matching time) is reduced from $5N$ to $3N$. Of course, with a speedup of $S$ or further time-multiplexing of these two combined channels together, a further reduction in delay will occur as well.

## VI. Simulation Results

In this section, we present the results of various simulations that we have performed to verify our theoretical results and observations in the previous sections.

In our first set of experiments, we consider a uniform traffic model where packets arriving to each input have a uniform distribution of output destinations. That is, the probability that a packet arriving at input $i$ has output destination $k$ is uniformly $1/N$. In this first set of experiments, we consider a Bernoulli i.i.d. arrival process. Using the CMS architecture, we compare results using three matching algorithms. The first is a self-randomizing matching algorithm called SERENA [14]. We will use this matching algorithm as the reference algorithm for the CMS architecture because it guarantees 100% throughput for all admissible Bernoulli i.i.d. traffic with no speedup, it has good delay properties, and it can be amortized to $O(1)$ complexity with sequential hardware for scalability[1]. The second is the widely used iSLIP [10] algorithm. We have included iSLIP for comparisons because it is often used as a reference matching algorithm for performance. Although widely used and effective, it should be noted that it requires parallel hardware, which means that the amount of processing hardware per linecard in the CMS architecture would be directly dependent on $N$, which limits scalability. The third is the maximum weighted matching (MWM) algorithm, which is known to achieve 100% throughput without speedup and very good average delays. Though impractical to implement at high speeds, we have also included it for comparisons.

[1]Note that SERENA is *not* a maximal matching algorithm and does not guarantee maximal matchings. In [14], a version of SERENA called MAX-SERENA was proposed to greedily derive a maximal matching by attempting to match unmatched inputs. However, to retain an $O(1)$ amortized complexity implementation, we did not implement this extension.

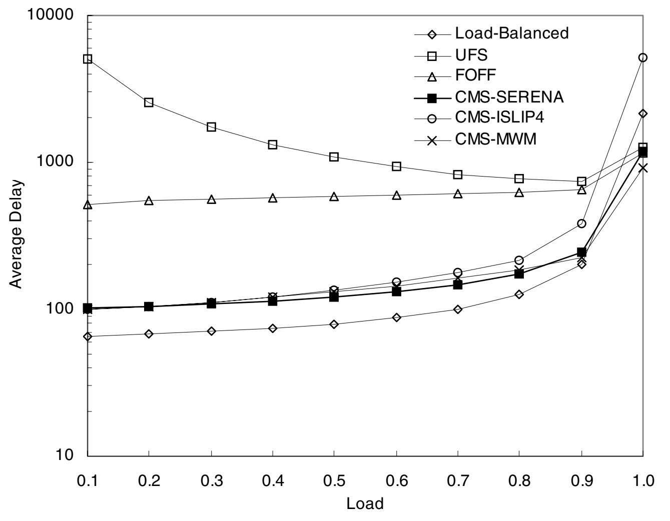

Fig. 5. Average delay under the uniform Bernoulli traffic model. Switch size is $N = 32$.

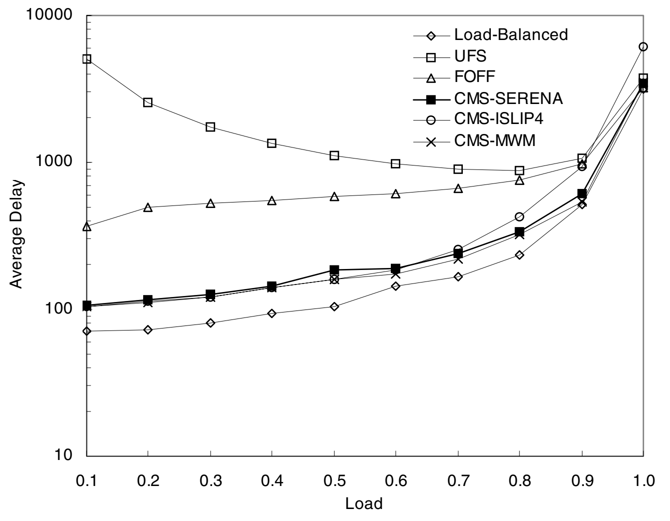

Fig. 6. Average delay under the uniform Pareto traffic model. Switch size is $N = 32$.

In addition to results using different matching algorithms with the CMS architecture, we have also included in this first set of experiments simulation results on average delay under the uniform Bernoulli traffic model for the originally-proposed load-balanced switch with no packet ordering guarantees [1], the frame-aggregation based method called *uniform frame spreading* (UFS) [4], and the frame-aggregation based method called *full-ordered frame first* (FOFF) [4].

Simulation results for this first set of experiments are shown in Figure 5. Several observations can be made in this first set of experiments.

- First, the average delay of a CMS with SERENA is about the same as a CMS with MWM under the uniform Bernoulli traffic model, even though SERENA is much less complex to implement than MWM. This demonstrates that the CMS architecture can achieve good results using an $O(1)$ amortized time matching algorithm. SERENA also performs better than iSLIP when used in the CMS architecture, especially under heavy load.
- Second, the average delay of a CMS with SERENA under uniform Bernoulli traffic is about the same as the basic load-balanced switch. For instance, as explained before, at light loads CMS requires a propagation delay of some $3N$, while the basic load-balanced switch requires some $2N$. However, unlike the basic load-balanced switch that can badly mis-sequence packets, the CMS architecture guarantees packet ordering and does not require an additional delay to reorder packets.
- Third, as expected, UFS incurs a high average packet delay under light load because of the need to accumulate full frames.
- Finally, although the average delay of the CMS architecture converges to the average delays of UFS and FOFF as the load $\rho$ approaches to $1.0$, the average delay of the CMS architecture is much lower for load $\rho \leq 0.9$. Therefore, low average packet delays can be achieved with only moderate speedup.

The same trends can be seen for different switch sizes.

In the next set of experiments, we consider average delays in response to bursty traffic. Instead of a Bernoulli i.i.d. arrival process, packets arrive in bursts. In particular, we ran simulations using random burst lengths that are chosen independently using the following (truncated) Pareto distribution:

$$\beta(i) = \frac{c}{i^{2.5}}, \text{ for } i = 1, \ldots, 10,000,$$

where $\beta(i)$ is the probability that a burst length of length $i$ is chosen, and

$$c = \left( \sum_{i=1}^{10,000} \frac{1}{i^{2.5}} \right)^{-1}$$

is the normalization constant. Using this Pareto distribution, the burst lengths can vary from 1 to $10,000$ packets. We again consider a uniform traffic model where packets arriving to each input have a uniform distribution of output destinations. The simulation results for this set of experiments are shown in Figure 6. In this set of experiments, we can make the following observations.

- First, we again see that the performance of the CMS architecture with SERENA and MWM are about the same, and the performance of a CMS with SERENA is better than the performance with iSLIP.
- Second, the performance of a CMS architecture is comparable to a basic load-balanced switch without packet ordering guarantees.
- Third, UFS incurs a high average packet delay under light load.
- Finally, although the average delay of the CMS architecture converges to the average delays of UFS and FOFF as the load $\rho$ approaches to $1.0$, the average delay of the

CMS architecture steadily declines under bursty traffic as load decreases. Therefore, the performance of a CMS architecture can be improved with speedup.

We observe the same trends for different switch sizes[2].

## VII. Load-Balanced Scheduling for Crossbars

We briefly explore in this section how some of the ideas from the CMS architecture can be applied to traditional single crossbar switch architectures. In particular, the CMS architecture has been designed to address two sources of bottlenecks in traditional centralized crossbar-based architectures. First, the use of fixed configuration meshes that are amenable to optics addresses the scalability limitations of traditional electronic crossbars that require frequent reconfigurations. It has been shown in [3] that the use of fixed configuration meshes in optics can be scaled to very high line rates and port counts. Second, although the CMS architecture also uses matching algorithms for conflict resolution, it reduces the complexity of the matching algorithm used by *load-balancing* the scheduling problem across $N$ *slower* schedulers. This second idea is also applicable to improving the practicality of traditional crossbar-based architectures. In particular, for moderate size high-performance switches with relatively small port counts, electronic crossbar fabrics can be made fast enough. However, the performance of these crossbar switches is often limited by the performance of the centralized scheduler. For example, MWM is known to be very effective, but its algorithmic complexity is prohibitive. We can apply the *load-balanced scheduling* idea developed for the CMS architecture to help alleviate the time complexity of centralized schedulers by *load-balancing* the work across $K$ slower schedulers, and the time complexity of the scheduling algorithm used would be amortized by a factor of $K$. $K$ does not need to be equal to $N$ – it can be chosen to provide the necessary reduction in time complexity. All the theoretical results developed in Sections III and IV and proved in the appendices directly apply.

## VIII. Conclusions

In this paper, we proposed the concurrent matching switch as a scalable two-stage switch architecture that guarantees packet ordering. From a scalability perspective, the concurrent matching switch architecture uses the same two stages of fixed uniform meshes as in current load-balanced switch architectures. These fixed uniform meshes do not require arbitrary per-packet switch configurations and are amenable to scalable implementation in optics. To enforce packet ordering throughout the switch, the proposed architecture uses a novel scalable distributed contention resolution approach where each linecard in the architecture independently solves a local bipartite matching problem that requires only local state information. The proposed architecture can then leverage the large, well-developed, and still progressing, body of work on scheduling algorithms to solve the matching problem. In particular, we showed that the concurrent matching switch can achieve 100% throughput guarantees using any stable matching algorithms, including practical algorithms that do not require speedup. We showed that each linecard has $N$ time slots to perform each matching step, and therefore the complexity of current matching algorithms can be amortized by a factor of $N$. Specifically, we showed that a class of provably stable matching algorithms with good delay properties can be amortized to $O(1)$ complexity using only sequential hardware. Therefore, the amortized complexity of the matching step can be made independent of the switch size. Finally, we showed that good average delays compared to existing load-balanced switch architectures can be achieved.

In a way, the concurrent matching switch architecture tries to combine the advantages of load-balanced routers with the well-developed body of work on matching algorithms for contention resolution to achieve a new scalable solution that exploits the advantages of both classes of architectures. From this perspective, we believe that this work enables router companies that have invested significant resources in developing a matching algorithm with good throughput guarantees to scale their routers by combining several routers together without losing their throughput guarantee, the packet order, or more simply their algorithm along the way. Moreover, this novel architecture opens the possibility for a great deal more research in this direction.

## IX. Acknowledgments

The authors would like to acknowledge the support of the ATS-WD Career Development Chair.

## References

[1] C. S. Chang, D. S. Lee, Y. S. Jou, "Load balanced Birkhoff-von Neumann switches, Part I: one-stage buffering," *Computer Communications*, vol. 25, pp. 611-622, 2002.

[2] C. S. Chang, D. S. Lee, C. M. Lien, "Load balanced Birkhoff-von Neumann switches, Part II: multi-stage buffering," *Computer Communications*, vol. 25, pp. 623-634, 2002.

[3] I. Keslassy, S. T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling Internet routers using optics," *ACM SIGCOMM*, Karlsruhe, Germany, 2003.

[4] I. Keslassy, "The Load-Balanced Router," *Ph.D. Thesis*, Stanford University, 2004.

[5] C. S. Chang, D. S. Lee, Y. J. Shih, "Mailbox switch: a scalable two-stage switch architecture for conflict resolution of ordered packets," *IEEE INFOCOM*, Miami, FL, 2004.

[6] B. Lin, I. Keslassy, "A scalable switch for service guarantees," *Proceedings of the 13th IEEE Symposium on High-Performance Interconnects*, Aug 17-19, 2005.

[7] C. S. Chang, W. J. Chen, H. Y. Huang, "On service guarantees for input buffered crossbar switches: a capacity decomposition approach by Birkhoff and von Neumann," *IEEE IWQoS'99*, pp. 79-86, London, UK, 1999.

[8] J. G. Dai, B. Prabhakar, "The throughput of data switches with and without speedup," *IEEE INFOCOM*, 2:556-564, Tel Aviv, Israel, March 2000.

[9] E. Leonardi, M. Mellia, F. Neri, M. A. Marsan, "On the stability of input-queued switches with speed-up," *IEEE/ACM Transactions on Networking*, vol. 9, no. 1, pp. 104-118, 2001.

[10] N. McKeown, "Scheduling algorithms for input-queued cell switches," *Ph.D. Thesis*, University of California, Berkeley, 1995.

[11] Y. Li, S. Panwar, H. J. Chao, "On the performance of a dual round-robin switch," *IEEE INFOCOM*, pp. 1688-1697, 2001.

[2]Though not shown due to space limitations, results have been obtained for CMS-SERENA with $N = 128$, and the same trends can be observed with respect to the basic load-balanced switch, UFS, and FOFF.

[12] N. McKeown, V. Anantharan, J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE INFOCOM*, vol. 1, pp. 296-302, San Francisco, CA, March 1996.

[13] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," *IEEE INFOCOM*, vol. 2, pp. 533-539, New York, 1998.

[14] P. Giaccone, B. Prabhakar, D. Shah, "Randomized scheduling algorithms for input-queued switches," *IEEE Journal of Selected Areas in Communication*, vol. 21, No. 4, pp. 642-655, May 2003.

[15] E. Leonardi, M. Mellia, F. Neri, M. Ajmone Marsan, "Bounds on delays and queue lengths in input-queued cell switches," *Journal of the ACM*, Vol. 50, No. 4, pp. 520-550, July 2003.

[16] S. Iyer, N. McKeown, "Analysis of the Parallel Packet Switch Architecture," *IEEE/ACM Transactions on Networking*, vol. 11, no. 2, April 2003.

## Appendix I
## Proof of Theorem 1

Theorem 1 states that if the matching algorithm applied on the request tokens at each intermediate input is strongly stable, then the CMS architecture is strongly stable as well.

We want to study the arrival process of token requests at each of the $N$ intermediate input linecards. In order to do so, we will first define a new time reference that is internal to each intermediate input. At each intermediate input $j$, tokens can only be received and granted (respectively packets can only arrive and depart) *every $N$ time-slots* (Equation (2)). Therefore, at each intermediate input, we will cut time into frames of $N$ time-slots. We will denote the new frame-based time by the symbol $t$ (where $t$ is the frame number, while $n$ is the time-slot number). Also, for each function $f$, when there is no confusion possible, we will use $f(t)$ for $f(n)$ with $n = Nt$, so as to avoid introducing new notations.

*Proof: (of Theorem 1)* Consider some intermediate input $j$. We'll show that the traffic it receives at each frame period has the same distribution as the traffic it would receive during a time-slot if the arrival traffic had a rate matrix $\Lambda$.

First, at each frame period, each input $i$ is connected with intermediate input $j$ *exactly once*, during a single time-slot (Equation (2)). During this time-slot, it receives at most one packet according to the Bernoulli i.i.d. process defined above using arrival matrix $\Lambda$, and converts this packet into a request token. Therefore, the request token arrival traffic to intermediate input $j$ during any frame-slot follows *exactly* the same distribution as the packet arrival traffic to the router during any time-slot.

Consequently, since the matching algorithm is strongly stable (Definition 1), at each intermediate input $j$

$$\limsup_{n\to\infty} E\left[\sum_{i,k} Q_{ijk}(n)\right] < \infty. \tag{25}$$

But as defined in the architecture model, the number of packets queued in the switch is

$$\begin{aligned} X_{ijk}(n) &= (A_{ijk}(n)) - (D_{ijk}(n)) \\ &= (R_{ijk}(n+N-1)) - (G_{ijk}(n-(3N-1))) \\ &= [R_{ijk}(n+N-1) - R_{ijk}(n-(3N-1))] \\ &\quad +[R_{ijk}(n-(3N-1)) - G_{ijk}(n-(3N-1))] \\ &\leq [N-1+3N-1]*N + Q_{ijk}(n-(3N-1)) \\ &= N(4N-2) + Q_{ijk}(n-(3N-1)), \end{aligned} \tag{26}$$

where we use the fact that at most $N$ packets (i.e., request tokens) can arrive to the switch at each time-slot. Consequently, we also have

$$\limsup_{n\to\infty} E\left[\sum_{i,j,k} X_{ijk}(n)\right] < \infty, \tag{27}$$

which proves the strong stability of the CMS architecture. ∎

## Appendix II
## Proof of Theorem 2

*Proof: (of Theorem 2)* The proof using speedup is *exactly* the same as the one without speedup. In fact, when using speedup, consider internally replacing *time-slots* by *phases*. All the properties of the internal components of the switch are exactly the same, and likewise the same conclusion follows. We do not repeat all the equations for the sake of clarity. ∎

## Appendix III
## Proof of Theorem 3

Theorem 3 states that if the matching algorithm applied on the request tokens at each intermediate input is stable, then the CMS architecture with flow-splitting is stable as well. To prove this, we will first need to demonstrate that request tokens are effectively distributed among intermediate inputs as packets arrive and are not getting stuck at the request token queues of the input linecards (Lemma 1). Then, we will show that the arrival process of token requests at each of the $N$ intermediate inputs follows some admissible average rate matrix (Lemma 2). This will enable us to use the stability properties of the matching algorithms and conclude with the proof of Theorem 3.

*Lemma 1 (Request Token Queue Size):* The size of a request token queue cannot exceed $N$.

*Proof:* Since request tokens are load-balanced in a round-robin way among intermediate inputs, starting with intermediate input 1, References [2], [4] show that

$$A_{ijk}(n) = \left\lceil \frac{A_{ik}(n)+1-j}{N} \right\rceil. \tag{28}$$

Moreover, at most one packet arrives to any input linecard at any time-slot, and at most one request token is consequently created. Using these two assumptions, Theorem 6 of [16] shows that the arrivals to token queue $R_{i,j}$ are bounded by a leaky bucket source of average rate $\rho = R/N$ and burst size $\sigma = N$ while it is periodically serviced at a rate $\mu = R/N$, and therefore a FIFO queue of size $N$ is sufficient. (Lemma 3 of [2] has a similar proof.) ■

We will now show that using the internal time definition based on frames, the arrival process of token requests at each of the $N$ intermediate inputs follows some admissible average rate matrix.

*Lemma 2 (Request Token Arrivals):* Assume that the packet arrival process to the router follows some admissible average rate matrix $\Lambda$ (Equations (22) and (23)). Then at each intermediate input, using the internal time definition, the request token arrival process also follows the same admissible average rate matrix $\Lambda$.

*Proof:* As defined above, $R_{ijk}(n)$ denotes the cumulative number of request tokens for flow $(i, j, k)$ arrived to intermediate input $j$ by the end of time-slot $n$. Using Lemma 1 and Equation (2), we find that a request token queued at time $n$ leaves its input linecard by time-slot $n + N^2$ and arrives to its intermediate input linecard by the end of time-slot $n + N^2 + (N - 1)$. Moreover, a request token cannot obviously arrive to the intermediate input before it is created at the input. Therefore, we get the double inequality

$$A_{ijk}(n - (N^2 + N - 1)) \leq R_{ijk}(n) \leq A_{ijk}(n). \tag{29}$$

Further, after dividing by $n$, we also have

$$\frac{A_{ijk}(n - (N^2 + N - 1))}{n} \leq \frac{R_{ijk}(n)}{n} \leq \frac{A_{ijk}(n)}{n}. \tag{30}$$

Moreover, using Equation (28),

$$\frac{A_{ik}(n)}{N} - 1 \leq A_{ijk}(n) \leq \frac{A_{ik}(n)}{N} + 1, \tag{31}$$

and from Equation (22), w.p. 1

$$\lim_{n\to\infty} \frac{\frac{A_{ik}(n)}{N}}{n} = \frac{\lambda_{ik}}{N}. \tag{32}$$

Therefore, combining these results, w.p. 1

$$\lim_{n\to\infty} \frac{R_{ijk}(n)}{n} = \lim_{n\to\infty} \frac{A_{ijk}(n)}{n} = \frac{\lambda_{ik}}{N}. \tag{33}$$

We can now use the internal frame-slot defined above. $N$ time-slots correspond to one frame, therefore w.p. 1

$$\begin{aligned} \lim_{t\to\infty} \frac{R_{ijk}(t)}{t} &= \lim_{t\to\infty} N \cdot \frac{R_{ijk}(Nt)}{Nt} \\ &= N \cdot \lim_{\substack{n\to\infty \\ n=Nt}} \frac{R_{ijk}(n)}{n} \\ &= N \cdot \frac{\lambda_{ik}}{N} \\ &= \lambda_{ik}. \end{aligned} \tag{34}$$

By definition, $[\lambda_{ik}]$ is doubly sub-stochastic as well, and therefore the proof of Equations (22) and (23) is completed. ■

We can now prove Theorem 3 about the stability of the CMS architecture with flow-splitting.

*Proof: (of Theorem 3)* The above Lemma 2 shows that at each intermediate input, the request token arrival process follows the admissible average rate matrix $\Lambda$ when using the internal time definition. Therefore, by Definition 2 of algorithm stability, the grant token arrival process at any intermediate input linecard follows the admissible average rate matrix $\Lambda$ as well when using the internal time definition. In other words, w.p. 1

$$\lim_{t\to\infty} \frac{G_{ijk}(t)}{t} = \lambda_{ik}. \tag{35}$$

We now translate this result in frame-slots into a result in time-slots. Using the transformation $n = Nt$ as in the above Equation (34), we get w.p. 1

$$\lim_{\substack{n\to\infty \\ n=Nt}} \frac{G_{ijk}(n)}{n} = \frac{\lambda_{ik}}{N}. \tag{36}$$

But each function $G_{ijk}(n)$ is non-decreasing, and therefore we will now prove that the above limit can be extended to time-slots that are not multiples of $N$. For instance, for each time-slot $n$, let $t(n)$ be such that $Nt(n) \leq n < N(t(n) + 1)$. Then by monotonicity of $G$

$$G_{ijk}(Nt(n)) \leq G_{ijk}(n) \leq G_{ijk}(N(t(n) + 1)), \tag{37}$$

which can be rewritten as

$$\begin{aligned} \frac{G_{ijk}(Nt(n))}{Nt(n)} \cdot \frac{Nt(n)}{n} &\leq \frac{G_{ijk}(n)}{n} \\ &\leq \frac{G_{ijk}(N(t(n) + 1))}{N(t(n) + 1)} \cdot \frac{N(t(n) + 1)}{n}. \end{aligned} \tag{38}$$

By Equation (36), both the above left and right expressions converge to $\lambda_{ik}/N$ w.p. 1 as $n$ goes to infinity, and therefore the middle one as well: w.p. 1

$$\lim_{n\to\infty} \frac{G_{ijk}(n)}{n} = \frac{\lambda_{ik}}{N}. \tag{39}$$

Since packet departures directly correspond to grant token generations with a fixed delay (Equation (15)), w.p. 1

$$\lim_{n\to\infty} \frac{D_{ijk}(n)}{n} = \frac{\lambda_{ik}}{N}. \tag{40}$$

Finally, summing up over all $N$ intermediate input linecards, we obtain w.p. 1

$$\lim_{n\to\infty} \frac{D_{ik}(n)}{n} = \lambda_{ik}, \tag{41}$$

which is exactly the definition of stability. ■

## APPENDIX IV
## PROOF OF THEOREM 4

*Proof: (of Theorem 4)* As in the proof of Theorem 2, the proof using speedup is *exactly* the same as the one without speedup. The only change is an *external* property: we know that that at most one packet arrives to every input at every time-slot. In other words, at most one packet arrives to every input at every $S$ phases. This is a less constraining property, and as a result the request token queue size can be reduced by a factor $S$ (new leaky bucket constraint in the proof of Lemma 1). All the following results are the same (inequalities are simply tightened), and likewise the same conclusion follows. ■