Collector-based Cell Reordering in Load-Balanced Switch Fabrics

Spyridon Antonakopoulos
Google, 76 Ninth Avenue, New York, NY, 10011
spyros@google.com

Steven Fortune, Rae McLellan, Lisa Zhang
Bell Laboratories, 600 Mountain Ave, Murray Hill, NJ 07974
firstname.lastname@alcatel-lucent.com

***Abstract*—Load-balanced switch fabrics offer the promise of very high capacity without the requirement of increased operating rates. We provide a novel solution to the well-known cell reordering problem, which arises in load-balanced switch fabrics if different paths through the switch fabric have different delays. We give a simple sorting circuit, easily implemented in hardware, that can be placed in the final stage of the fabric. This minimizes the delay through the fabric and removes any constraints on operation of the first stage. We show that the sorting circuit can be implemented to operate at a rate of one cell per cycle; we give a probabilistic analysis of required queue occupancies, both at the sorting circuit and at midstage elements; we also briefly discuss congestion control in load-balanced switch fabrics.**

I. Introduction

Load balancing in switch fabrics has received attention as a way to obtain very high capacity routers [1], [2], [3], [4], [8], [9], [10], [12]; see [9], [10] for an introduction. Continued attention is warranted by the continuing rapid increase in Internet traffic, variously estimated at 30-50% per year [5], contrasting with slow current increase in VLSI clock rates. Very soon every data path in a switch fabric will need to handle one data cell per clock cycle (at a 1Tbps line rate, a 512 bit cell arrives every 500ps, close to the feasible cycle time of SRAM). Load balancing provides parallel data paths and hence increased capacity.

A load-balanced switch fabric consists of three stages of cell-handling elements, which we call *distributors*, *routing elements*, and *collectors* (see Figure 1). A distributor is part of the input side of a router line card. The line card splits arriving data packets into a stream of fixed-size cells, which the distributor distributes approximately uniformly to the (midstage) routing elements. A routing element performs routing by sending each cell to its intended collector. A collector is part of the output side of a router line card; cells that arrive at a collector must be reassembled into packets to be emitted by the line card.

Load balancing in this fashion enables very high capacity switch fabrics: no central scheduling algorithm is required, and the individual cell-handling elements need only operate at about the rate that traffic can arrive at a distributor. With m elements at each stage the aggregate fabric capacity is increased by a factor of m over the per-element rate.

A well-known disadvantage of load-balanced switch-fabrics is the problem of cell reordering. Cells from a specific distributor sent to a specific collector traverse differing routing

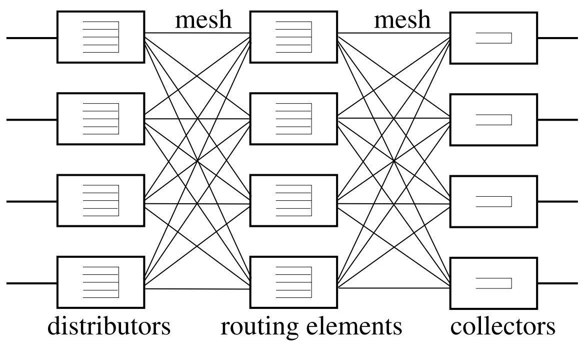

Fig. 1. Load-balanced switch fabric

elements. If the routing elements have differing delays (e.g. because of differing queue occupancies), cells may arrive at the collector in a different order from which they were sent, signficantly complicating the reassembly of packets from cells.

Many solutions to the reordering problem have been proposed in the literature [3], [4], [8], [9], [10], [12]; see [8], [12] for surveys. In general, the solutions impose constraints on how cells are sent from the distributor. For example, FOFF [9], [10] requires that cells be sent in a frame, i.e. in parallel to all routing elements simultaneously, and that the frame be uniform, i.e. all cells in the frame have the same destination. Unfortunately, this introduces delay to fill out a uniform frame. With FOFF average cell delay is bounded by an amount quadratic in the number of distributors, but the worst-case delay is unbounded. Other solutions similarly introduce delay. In addition, constraining distributor behavior makes it problematic to accomodate other router requirements, such as multicast traffic and multiple levels of cell priority.

We give an alternate solution to the cell-reordering problem that imposes no constraints on distributor behavior and imposes no delay on cell departure: we place a sorting circuit in the collector. The circuit is essentially an implementation of parallel merge sort on streams using systolic arrays. It guarantees that for each distributor-collector pair, cells leave the collector in the same order that they were sent from the distributor; of course cells from different distributors may be intermixed as they leave the collector. A downstream process, not elaborated here, must reassemble cells into packets. The sorting circuit is conceptually quite simple; however, some details require careful elaboration. For example, the circuit must stall, i.e. stop emitting cells, if some intermediate cell has not yet arrived; conversely, the circuit must flush stored cells when it can be ascertained that there are no missing cells. We give a detailed argument showing that the circuit can operate at the rate of one cell per cycle; the circuit is readily implementable in VLSI.

The sorting circuit requires queues to store arriving cells; in addition queues are required in each routing element. We give a rigorous probabilistic analysis of the occupancy of these queues. For this we assume that traffic arriving at the switch fabric is "stochastically admissible", i.e. so that no collector is overloaded in expectation. We also assume that the throughput of the switch fabric is slightly larger than the arrival rate, by a factor of $\alpha > 1$. Our analysis gives tail estimates: we show that the probability that routing-element queue occupancy exceeds l is at most roughly $1/\alpha^l$. Collector queues have a similar analysis.

We remark that while we explicitly assume a fabric speedup, this assumption is implicit in prior analyses of load-balanced fabrics. For example, Chang *et al* [2] show 100% throughput for a similar model of load-balanced fabrics. By this they mean that average routing-element queue length grows proportionally to $1/\rho$, where the traffic arrival rate is less than capacity by a factor $\rho < 1$. Clearly this condition is equivalent to a fabric speedup of $\alpha = 1/\rho$. Without the assumption of speedup, queues would be unstable, i.e. grow unboundedly large with probability 1.

Of course, there is no guarantee that traffic is admissible. Since distributors do not coordinate, there may be *hotspot* traffic to a collector, i.e. the distributors jointly send more traffic to the collector than its departure capacity. This will cause routing-element and collector queues to grow, so some form of congestion control is required to signal distributors to stop sending cells. We briefly discuss congestion in Section V; further discussion and simulation will appear in the full paper.

Section II below discusses our model of load-balanced switch fabrics in more detail. Section III presents the traffic model and analysis of routing-element queue lengths. Section IV discusses the sorting circuit and collector queue lengths. Section V briefly examines congestion signaling. A few proofs appear in Section VI and the rest in the full paper.

II. LOAD-BALANCED SWITCH FABRICS

We now describe our model of a load-balanced switch fabric in more detail. A block diagram of the fabric appears in figure 1. It consists of m *distributors*, m midstage *routing elements*, and m *collectors* with two meshes of m^2 links, one connecting each distributor to each routing element, and one connecting each routing element to each collector.

All the stages operate in synchrony, with a master clock determining a *frame interval*. Each distributor receives a stream of fixed-size *full* cells, each containing a portion of a packet as payload and each labeled with a destination collector. Every frame interval, each distributor sends out one cell per link in parallel to all routing elements. A distributor can create *empty* cells with dummy payload, so that every link gets a cell.

Every frame interval, a routing element receives m cells in parallel, one from each distributor. A routing element has m queues, one per collector; arriving cells are placed in the appropriate queue, and empty cells are discarded. Every frame interval, the routing element sends out one cell per collector, where the cell sent to the kth collector is the head of the kth queue. Again, if a queue is empty, an empty cell is sent instead.

A collector receives up to m full cells simultaneously; empty cells are discarded. Cells pass through a sorting circuit (Section IV) and are then placed into a *reassembly queue*, prior to reconstruction of packets from cells.

A *frame* is the set of cells sent simultaneously, either from a distributor or a routing element. We assume that a *frame time* that counts frame intervals is available globally.

We remark that load-balanced switch fabrics are often described with each mesh replaced by a cross-bar operating with a fixed pattern of m matchings, where the tth matching, $t = 1, \ldots, m$, pairs input i with output $(i + t) \bmod m$. The difference between the two models is not substantive as long as a frame interval corresponds to the time required for m matchings: in the frame-based model m cells are sent in parallel every frame interval; in the cross-bar-based model m cells are sent one-by-one during the frame interval. The frame-based model is convenient for the analysis in Sections III and IV below.

A. Technology

A load-balanced switch fabric is of most interest as a way of obtaining very high bandwidth. We briefly discuss technology issues that determine switch-fabric parameters. These issues will be relevant to the discussion in Section IV.

The clock rate of a VLSI circuit is usually determined by the cycle time of SRAM. Within one cycle, a location of SRAM can be read or written, or data can be moved from one register to another possibly through some combinational logic. We assume that distributors, routing elements, and collectors all can operate at an internal rate of one cell per cycle. Hence a frame interval must be at least m cycles, since m cells arrive every such interval; we assume a frame interval is exactly m cycles.

Since we assume a fabric speedup $\alpha > 1$, cell size s must be

$$s := \alpha C \tau,$$

where C is the maximum data rate in bits per second arriving at a distributor or leaving a collector and τ is the cycle time of SRAM.

We assume that the mesh links operate at a data rate L much less than C. The number of links m entering or leaving a switch-fabric element is then

$$m := \alpha C / L$$

to ensure an aggregate bandwidth of αC. The frame interval F is now both m cycles and the time to send a cell at rate L:

$$F = s/L = m\tau.$$

Notice also that C bandwidth translates to a maximum arrival or departure rate at distributor or collector of

$$r = m/\alpha$$

cells within a frame interval.

Nominal values for these parameters are $C = 1$ Tbps, $\alpha = 1.28$, $\tau = 400$ ps, and $L = 10$ Gbps. These yield a cell size $s = 512$ bits; the number of distributors, routing elements, and collectors as $m = 128$; a frame interval $F = 51.2$ ns; and a maximum arrival and departure rate of $r = 100$ cells per frame interval.

III. Traffic model and queue lengths

We now introduce the traffic model that we use in our theoretical analysis and present probabilistic bounds on the sizes of routing-element queues. The model will also be used in the analysis of Section IV. We remark again that most formal proofs are deferred until Section VI or the full paper.

An arrival process determines the cells that arrive at each distributor during each frame interval. The process is *stochastically admissible* if the arrival distributions at distinct frame times and at distinct distributors are independent, and if at every frame time the expected number of cells destined for a specific collector is at most r. As examples, suppose possibly time-varying numbers $\rho_{ik} \geq 0$ with $\sum_i \rho_{ik} \leq 1$ are given. The process where each cell arriving at distributor i has destination collector k with probability ρ_{ik} is stochastically admissible. So is the process where all r cells have the same destination collector k with probability ρ_{ik}.

The *random-consecutive* frame-filling strategy is the following: given $p \leq m$ cells $c_1, \ldots, c_p$, the distributor chooses a random integer t, places the full cells in slots $c_{t+1}, \ldots, c_{t+p}$, where the slot indices are taken mod m, and places empty cells elsewhere. This strategy suffices for the analysis below and has the advantage that it requires a random choice only once per frame interval.

Lemma 1. *Suppose the arrival process is stochastically admissible, routing-element queues are initially empty, and routing-element queues are unbounded (so congestion is never signaled). There is a constant $\kappa > 0$ (depending on α) such that at any frame time and for any collector:*

1) *If distributors use the random-consecutive strategy, the probability that any single routing-element queue contains more than l cells is at most κ/α^l.*
2) *The probability that the total number of cells in all routing-element queues exceeds rl is at most κ/α^l.*

Simulation shows routing queue lengths somewhat smaller than the bound in this lemma; for example with $\alpha = 1.28$, the probability that routing-element queue lengths exceed length l is at most about $.62^l$, whereas $1/1.28 \approx .78$.

IV. Reordering

As discussed above, cells may arrive at a collector out of order if routing-element queues have different occupancies. We now discuss how to reorder cells at the collector.

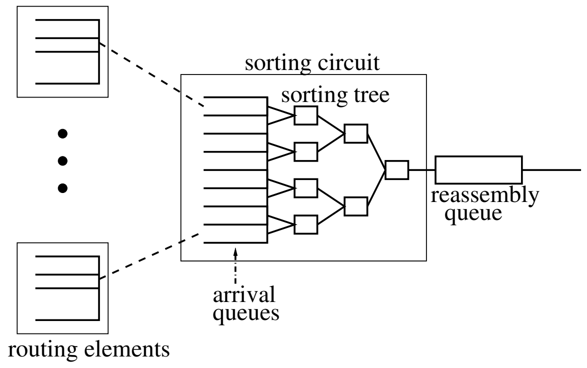

Fig. 2. Sorting tree

Consider the stream of cells arriving at the switch fabric with a specific collector as destination. Each distributor takes its arriving stream of packets and splits it into m streams, one per routing element. Each routing element merges m incoming streams, and then sends the resulting stream to the collector. The task for the collector is to merge its m input streams into a single stream to be sent to the reassembly queue, with the requirement that the cells arriving from a specific distributor are in the same order as sent from the distributor.

This task can be accomplished with what amounts to a merge-sort on streams. Each cell is annotated with a key consisting of three fields: the frame time that the cell left the distributor, the index of the distributor, and the arrival index of the cell within its frame, in order from most significant to least significant. This three-field key provides a total linear order on cells; a cell c *precedes* a cell c', $c \prec c'$, if the key of c is lexicographically less than the key of c'. Whenever streams are merged, the linear order should be maintained. In consequence, when cells are sent to the reassembly queue at the collector, all cells with a particular frame time will appear together as a block; within this block all cells from a specific distributor will appear together as a subblock; and within a subblock they are ordered by arrival at the distributor. (We remark that interchanging the significance of distributor index and arrival index in the lexicographic comparison would also work, with an altered output order).

Maintaining the linear order is easy at a routing element. At each frame time, all arriving cells have the same frame time and at most one cell arrives from each distributor, so the only requirement is that cells be added to queues ordered by distributor index.

A. Sorting circuits

Maintaining order at the collector can be accomplished by the insertion of a *sorting circuit* ahead of the reassembly queue. The sorting circuit consisting of m arrival queues and a sorting tree, as shown in Figure 2.

Each arrival queue is associated with an upstream routing-element queue and stores cells temporarily as they arrive from the upstream queue. Logically, the sorting tree is a binary tree. The leaves of the sorting tree are the arrival queues. An internal tree node has two inputs, received from its children, and one output, sent to its parent. Each internal node stores a single cell, and has access to the cells of its two children. A cell may be full or empty, i.e. with or without payload, but in either case the cell has a sort key. The sorting tree is balanced, assuming for simplicity that m is a power of two. The tree produces a single sorted stream of cells from the cells in the arrival queues, as we now describe.

Every cycle, a cell is requested from the root of the tree. When a cell is requested from a node, the node responds with its stored cell. To replace the cell, the node compares the keys of the cells of its two children, and requests the node with the smaller key to send its cell, which becomes the new stored cell. If the two children are arrival queues, the keys of the corresponding queue heads are compared, and the smaller is deleted from its queue and becomes the new stored cell. The cell provided by the root of the tree is sent to the reassembly queue, as long as the cell is not empty.

To ensure correct operation of the sorting circuit, two issues need to be addressed. First, we must ensure that cells are emitted in sorted order; in particular, we must ensure that no cell is emitted from the sorting tree if a preceding cell has not yet arrived at the sorting circuit. Second, we must ensure that cells are eventually emitted from the tree, even if no further cells arrive.

For the first issue, we associate with each arrival queue a register to store an empty-cell key. At any time, this key should precede the key of any cell that can subsequently arrive at the arrival queue; we discuss below how this is to be done. If an arrival queue is empty, its parent bases its comparison on the empty-cell key instead of the head-of-queue key. If the parent requests a cell from an empty arrival queue, the arrival queue creates an empty cell whose key is the empty-cell key, and sends that cell to the parent. Similarly, if a parent of an interior node requests a cell from a child storing an empty cell, the empty cell is sent to the parent. Hence empty cells can propagate up the tree all the way to the root.

A full cell at a node is *blocked* if its sibling node stores an empty cell with smaller key. Clearly, while a cell is blocked, it is not sent to the parent of the node. Thus the purpose of empty cells is to maintain sorted order, as demonstrated in the following proposition.

Proposition 2. *The sorting tree emits cells in sorted order, as long as empty cells emitted from the root are discarded.*

Proof: It is easy to verify that the sorting tree maintains a slight variant of the heap property: the cell c stored at a node precedes every other full cell in the subtree rooted at the node, including all cells currently in arrival queues or that will arrive in the future; possibly some empty cell in the subtree has the same key as c. Hence the cell stored at the root of the tree precedes every full cell in the tree or that will ever arrive. ■

It remains to handle the second issue, ensuring that cells are eventually emitted, or equivalently that blocked cells eventually become unblocked. We require that a routing element only send an empty cell to a collector if the corresponding routing-element queue is empty. Thus if an empty cell does arrive, the collector can infer that any subsequent cell received from the same routing element will have frame time after the current frame. The arrival queue should maintain its empty-cell key as the larger of two values: the key of the last full cell sent to the arrival-queue parent, in which case the key is a *ghost* key, or a key with the frame time that was current when the last empty cell arrived from the routing element, in which case the key is an *(empty-frame) marker* key.

To make sure that no cell remains blocked, at every cycle, every internal node with an empty cell should generate a request to the child with smaller key to send its cell, which as usual replaces the empty cell. At the next frame time, a full cell may have arrived which will propagate up the tree. If not, an empty cell with a marker key will propagate up the tree, which necessarily has a key larger than any full-cell key in the tree. We prove in the full paper that any blocked cell will become unblocked in about a frame interval. This implies that all cells are eventually emitted, even with no subsequent arrivals of cells to the collector.

B. Collector arrival queue occupancy

The sorting circuit requires arrival queues to store cells until they can be admitted into the sorting tree. The required size of the arrival queues depends upon the past discrepancy of queue occupancies at routing elements. To see this, suppose a routing-element queue Q has l cells in it and the other routing-element queues for the same collector are empty. Suppose cell c arrives next at Q with other cells subsequently arriving at the other routing-element queues. In the l frame times before c moves to its collector arrival queue, each of the other routing-element queues could send a cell to each of the other collector arrival queues. None of these cells can leave the sorting circuit until c arrives at the collector, forcing arrival occupancy of approximately l. Since the sorting circuit drains at a fixed rate, occupancy can potentially be twice as large, as established by Lemma 3.

Lemma 3. *If δ bounds the maximum difference between routing-element queue occupancies at any time, then the sorting-circuit arrival-queue occupancy is at most $2\delta + 1$.*

Keslassy *et al* [10], [9] give a related result. Assuming routing-element queue occupancies differ by at most m, they show that a cell can always be removed from some arrival queue as long there is an arrival queue with occupancy m. Hence if cells can be removed instantaneously, arrival queue occupances never exceed m. The additional factor of 2 here results from the requirement that at most one cell is emitted from the sorting circuit per cycle.

Unfortunately, there is no worst-case bound on routing-element queue discrepancies; furthermore the probabilistic estimate of Lemma 1 together with Lemma 3 do not immediately give a probabilistic bound on arrival-queue lengths, since Lemma 3 bounds arrival queue occupancy on worst-case routing-element queue discrepancy over time. However, it is possible to obtain probabilistc arrival-queue occupancy bounds directly.

For the analysis, we assume that the tree can emit a cell from the root every cycle, with a frame time occuring at integer multiples of m. Sort-tree nodes operate in parallel and take a single cycle to move a cell from a child to a parent. Choosing the cells to move may require a computation along a path of length $\log_2 m$ from root to leaf (and indeed potentially along similar paths rooted at empty nodes). For now, we assume that this computation can happen within a cycle. This assumption is removed later, in Section IV-C.

For a cell c, let $a(c)$ be the time that the cell departs the routing-element queue and is added to an arrival queue, the transfer assumed to take no time; $a(c)$ is a multiple of m. Let $d(c)$ be the time that the cell departs the arrival queue and enters the tree. The number of cells in c's arrival queue at time $d(c)$ is at most $\lfloor (d(c) - a(c))/m \rfloor$, since any such cell must have arrived after c and cells arrive at most once per frame interval.

Theorem 4. *Suppose all queues are initially empty and the arrival process is stochastically admissible. For some constant κ' (independent of m and l but depending upon α), and for any cell c,*

$$\Pr((d(c) - a(c))/m \geq l) < \kappa' m^2 / \alpha^l.$$

To be explicit, if the arrival queue lengths are sampled when cells move from the arrival queue to the tree (i.e. at time $d(c)$ for each cell c), then the probability that the length exceeds l decreases exponentially with l, albeit with probability somewhat higher than for routing-element queue lengths.

C. Alternate sorting circuits

As described the sorting tree potentially requires synchronization of cell motion along a path from root to leaf. We describe an alternate sorting circuit requiring only local synchronization.

The alternate circuit has a tree where each node of the tree has two registers, each of which holds a cell, either full or empty. The tree should maintain the variant heap property, as before: both of the cells at a node should precede all the other cells in the subtree rooted at the node, including future arrivals in the arrival queue, though possibly some empty cell in the subtree has the same key as the later of the two cells.

Some consolidation of empty cells is possible since two cells are stored in a node. If two empty cells have distinct keys, the earlier key should be replaced with the later key; this maintains the variant heap property. Similarly, if a node contains a full cell and an empty cell, the key of the empty cell should be replaced with that of the full cell, if the empty cell had earlier key.

In operation, each node makes the key of its earlier cell available to its parent. Every cycle, every node storing an empty cell requests a cell from one of its children, as before. The requested cell obtained from the child replaces the empty cell. Simultaneously the node may be sending the cell with earlier key to its parent. These transactions can involve different registers, since either both registers have empty cells with the same key, or the register with full cell is sent to the parent, and the register with empty cell receives a cell from a child. Hence only synchronization between adjacent levels in the tree is required.

Lemma 5. *For any sequence of arrivals of cells to arrival queues, the alternate sorting circuit emits a full cell no later than would the original tree; similarly at any time any arrival queue in the alternate tree is at most as long as it would be in the original circuit.*

V. Congestion control

Since distributors choose cells to send independently of other distributors, it is possible that cells arrive at a collector at a rate faster than its departure capacity. In this case switch-fabric queue occupancy will grow, either at a collector reassembly queue or at a routing-element queue. A congestion-feedback signal is necessary to stop distributors from sending traffic to the collector until the corresponding switch fabric queues can drain.

Requisite queue sizes are in part determined by the response delay at a distributor to a congestion signal, i.e. the time required for the distributor to start or stop sending cells once signaled by a routing element or collector. The response delay may be substantially more than a frame time, perhaps tens of frame times, as a result of propagation delays, VLSI-level pipelining at the distributor, and DRAM access latency. Suppose the response delay is d frame times and congestion is signaled. In the worst case all m distributors could be sending all their cells to the same collector. Hence each routing-element queue needs at least dm cells of headroom. In fact, as discussed in the full paper, having multiple distributors sending traffic to the collector is typical of an overload situation. Similarly, as a queue empties, the congestion signal should be cleared roughly d frame times before the queue is empty, so that the queue maintains occupancy if traffic is available.

By Lemma 1 and Theorem 4, it is possible for queues to grow arbitrarily large even with stochastically admissible traffic. Such an event, a *false congestion* event, may cause congestion to be signaled. As discussed in the full paper, congestion signals may cause a slight loss in throughput to a collector The probability of such an event decreases exponentially in the queue size. Hence with sufficiently large queues throughput loss is negligible.

The sorting-circuit arrival queues do not need to use congestion signaling. Instead, if any arrival queue occupancy approaches the queue size, a backpressure signal to the upstream routing-element queue causes the routing-element queue to temporarily stop sending cells until the arrival queue drains. Assuming that a collector and a distributor for the same line card are colocated, the backpressure signal is easily implemented as an annotation bit on the cell sent from distributor to routing element, with a delay of a couple of frame intervals. Of course, the backpressure signal may cause the routing element queue to stop draining, and perhaps to grow, eventually signaling congestion.

VI. Proofs

A. Proofs from Section III

A random variable is *zero-one-bounded* if it can take a finite set of values in [0,1]. The following standard result on Chernoff bounds [11] is usually cited for zero-one random variables, but extends easily to zero-one-bounded variables.

Theorem 6. *Let $X_1, X_2, \ldots, X_n$ be independent zero-one-bounded random variables, $X = \sum X_i$, and $\mu = \mathbf{E}[X]$. Then*

$$\Pr(X \geq (1+\delta)\mu) < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}.$$

Proposition 7. *Suppose X_i are independent zero-one-bounded random variables, $X = \sum X_i$, and $\mathbf{E}[X] \leq f/\alpha$. For $l = 0, 1, 2, \ldots$ and $f = 1, 2, \ldots$,*

$$p_{fl} := \Pr(X \geq f + l) < F_{fl} < F_{f0}/\alpha^{l}$$

where

$$F_{fl} := \frac{e^{\frac{\alpha-1}{\alpha}f+l}}{(\alpha(1+l/f))^{f+l}}$$

Proof: Set $\mu_f = f/\alpha$ and $\delta_{lf} = \alpha - 1 + l\alpha/f$; hence $(1+\delta_{lf})\mu_{lf} = l+f$. The first inequality follows using Theorem 6, substituting δ_{lf} for δ and f/α for μ, and simplifying a bit.

For the second inequality, note that for $f = 1, 2, \ldots$

$$\frac{F_{fl}}{F_{f\,l-1}} = \frac{e}{\alpha}\frac{\left(1+\frac{l-1}{f}\right)^{f+l-1}}{\left(1+\frac{l}{f}\right)^{f+l}} < \frac{e}{\alpha}\left(\frac{f+l-1}{f+l}\right)^{f+l} < \frac{1}{\alpha}$$

where for the final inequality we use $(x/(x+1))^{x+1} < 1/e$ with $x = f + l - 1$. ■

Proposition 8. *Both $\sum_{f=1}^{\infty} F_{f0}$ and $\sum_{f=1}^{\infty} fF_{f0}$ are finite.*

Proof: $F_{f0} = (\frac{e^{(\alpha-1)/\alpha}}{\alpha})^{f}$; $e^{(\alpha-1)/\alpha} < \alpha$ follows easily by taking the natural logarithm of both sides and using $\alpha > 1$ and $\ln\alpha < \alpha - 1$. ■

Proof of Lemma 1: (1) Choose a routing-element queue for a specific collector. For a single frame interval, we let A_i be the zero-one random variable which is 1 if the ith distributor sends a cell to the specific routing element queue and 0 if not. Because the distributor uses the random-consecutive policy, the expected number of cells sent to the queue is a_i/m, where a_i is the number of cells arriving at distributor i destined for the specific collector. For distinct i, i', A_i and $A_{i'}$ are independent since the arrival distributions at distributors i and i' are independent. Summed over all distributors, the number of cells that get sent to the queue has expectation $\mathbf{E}[\sum A_i] = r/m = 1/\alpha$. Over f frame intervals, the number of cells sent to the specific queue is the sum of fm independent zero-one random variables, with expectation f/α. Hence Proposition 7 applies.

For simplicity we assume that a cell sent to a routing-element queue at time t can also depart to an arrival queue at time t (if there are no cells ahead of it in the queue). We measure queue occupancy after departures, so for example, if a queue is empty at time $t-1$ and a cell arrives at time t, it can also depart at time t and the queue is still empty at time t.

Suppose a routing element queue has occupancy $l > 0$. Consider the latest previous frame time when the queue was empty, say $f > 0$ frame times previous. During the next f frame times, the queue is nonempty, so a cell is sent to the collector every frame time. Hence $f+l$ cells must have arrived during the f frame times.

Thus, using a union bound, the probability at any time that the queue has l cells is bounded by $\sum_{f=1}^{\infty} p_{fl} < \sum_{f=1}^{\infty} F_{fl} < \sum_{f=1}^{\infty} F_{f0}/\alpha^{l}$, using Propositions 7 and 8.

(2) The proof is similar, using zero-one-bounded random variables whose value is the number of cells sent from a specific distributor, divided by r. ■

The proofs of other lemmas and theorems are omitted due to lack of space. The full paper with complete proofs is available from the authors.

References

[1] S. Antonakopoulos, S. Fortune, R. McLellan, L. Zhang, Worst-Case Delay Bounds for Uniform Load-Balanced Switch Fabrics, to appear, IEEE International Conference on Communications, 2013.

[2] C. Chang, D. Lee, Y. Jou, Load-balanced Birkhoff-von Neumann switches: Part I: One-stage Buffering, *Comput. Commun.* **25**:6, 2002, pp. 611-622.

[3] C. Chang, D. Lee, Y. Jou, Load-balanced Birkhoff-von Neumann switches: Part II: Multi-stage Buffering, *Comput. Commun.* **25**:6, 2002, pp. 623-634.

[4] C. Chang, D. Lee, Y. Shih, C. Yu, Mailbox Switch: A Scalable two-Stage Switch Architecture for Conflict Resolution of Ordered Packets, *IEEE Transactions on Communications*, **56**:1, January 2008, pp. 136–149.

[5] CISCO Visual networking Index: Forecast and Methodology, 2011-2016, Cisco, 2012.

[6] S.-T. Chuang, A. Goel, N. McKeown, B. Prabhakar, Matching Output Queueing with a Combined Input Output Queued Switch, Computer Systems Technical Report CSL-TR-98-758. March 1998, or *Proceedings of INFOCOM '99*, 1169-1178, IEEE, April 1999, or *IEEE Journal on Selected Areas in Communications*, **17**:6 December 1999, pp. 1030-1039.

[7] W. Feller, *An Introduction to Probability Theory and its Applications*, Volume I, Third Edition, John Wiley and Sons, NY, 1950.

[8] J. Jaramillo, F. Milan, R. Srikant, Padded Frames: a Novel Algorithm for Stable Scheduling in Load-Balanced Switches, *IEEE/ACM Transactions on networking*, **16**:3, October, 2006, pp. 1212–1225.

[9] I. Keslassy, The Load-balanced Router, Ph.D. dissertation, Stanford Univ, Stanford CA, 2004.

[10] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, N. McKeown, Scaling Internet Routers Using Optics (Extended Version), Stanford Technical Report TR03-HPNG-080101, 2003.

[11] M. Mitzenmacher, E. Upfal, *Probability and Computing: Randomized algorithms and probabilistic analysis*, Cambridge University Press, Cambridge, 2005.

[12] Y. Shen, S. Panwar, H. Chao, Design and Performance Analysis of a Practical Load-Balanced Switch, *IEEE Transactions on Communications* **57**:2, August 2009, pp. 2420–2429.