Efficient Buffering and Scheduling for a Single-Chip Crosspoint-Queued Switch

Zizhong Cao
Polytechnic Institute of NYU
5 MetroTech Center, Brooklyn, NY 11201
zcao02@students.poly.edu

Shivendra S. Panwar
Polytechnic Institute of NYU
5 MetroTech Center, Brooklyn, NY 11201
panwar@catt.poly.edu

ABSTRACT

The single-chip crosspoint-queued (CQ) switch is a self-sufficient switching architecture enabled by state-of-art ASIC technology. Unlike the legacy input-queued or output-queued switches, this kind of switch has all its buffers placed at the crosspoints of input and output lines. Scheduling is also performed inside the switching core, and does not rely on instantaneous communications with input or output line-cards. Compared with other legacy switching architectures, the CQ switch has the advantages of high throughput, minimal delay, low scheduling complexity, and no speedup requirement. However, since the crosspoint buffers are small and segregated, packets may be dropped as soon as one of them becomes full. Thus how to efficiently use the crosspoint buffers and decrease the packet drop rate remains a major problem that needs to be addressed. In this paper, we propose a novel chained structure for the CQ switch, which supports load balancing and deflection routing. We also design scheduling algorithms to maintain the correct packet order caused by multi-path switching. All these techniques require modest hardware modifications and memory speedup in the switching core, but can greatly boost the overall buffer utilization and reduce the packet drop rate, especially for large switches with small crosspoint buffers under bursty and non-uniform traffic.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*Packet-switching networks*; C.2.6 [**Computer Communication Networks**]: Internetworking—*Routers*

General Terms

Algorithms, Design

Keywords

Single-Chip, Crossbar, Load Balancing, Deflection Routing

ANCS'12, October 29–30, 2012, Austin, Texas, USA.

1. INTRODUCTION

In the past decade, modern Internet-based services such as social networking, video streaming and cloud computing have brought about a continuous, exponential growth in Internet traffic. The recent boom in smartphones, tablets and other portable electronic devices has made all these remote services more accessible to people, while imposing even larger traffic burdens on the backbone networks. To accomodate the increasing demands, the capability of Internet core switches must grow commensurately. Consequently, there has been interest in designing high-performance switching architectures and scheduling algorithms.

Many types of switching architectures have been proposed. The first kind is the output-queued (OQ) switch [24], in which an arriving packet is always directly sent to its destination output, and then buffered there if necessary. The OQ switch may achieve 100% throughput with infinite buffers, but requires an impractically high speedup. Specifically, the switching fabric of an $N \times N$ OQ switch may need to run N times as fast as the single line rate in the worst case, when all inputs target the same output.

Another popular kind of architecture is the input-queued (IQ) switch. In an IQ switch, packets are buffered at the input and served in a first-in-first-out (FIFO) manner if the target output is idle. IQ switches require no speedup, but suffer from the head-of-line (HOL) blocking problem, which limits the throughput to 58.6% [24]. This problem was later solved by implementing virtual output queues (VOQ) at each input. Various scheduling algorithms such as iSLIP [30], DRRM [28], and maximum weight matching (MWM) [36] have been proposed to achieve high throughput. However, many of these algorithms are complex, or require nearly instantaneous communications among input and output schedulers that are usually placed far apart on different line-cards. This might become a bottleneck for high-speed switches, in which the round-trip latency between different line-cards may span several time slots and thus is no longer negligible. For instance, the round-trip latency can be as high as about $100ns$ assuming $10m$ inter-rack cables, while each time slot lasts at most about $50ns$, assuming OC-192 or higher line speeds and $64byte$ fragmentation. A combination of IQ and OQ switches, the combined-input-and-output-queued (CIOQ) switch, has been proposed to achieve high throughput with minimal delay [13], but suffers from similar problems as an IQ switch.

In recent years, a new kind of structure called the buffered crossbar has been widely studied. Typically, one or a few buffers are placed at each crosspoint, while others are still placed at the inputs of a switch, which effectively becomes a combined-input-and-crosspoint-queued (CICQ) switch [31]. With the help of crosspoint buffers, scheduling becomes much easier for CICQ switches since input scheduling and output scheduling can now be performed separately. Many of the scheduling algorithms for IQ switches can

be directly applied to CICQ switches at a lower complexity, e.g., the distributed MWM algorithm DISQUO [39] and the push-in-first-out (PIFO) policy [14]. On the other hand, a CICQ switch suffers from the same problem as an IQ switch due to the need for nearly instantaneous control communications between the input line cards and the switching core. Kanizo et al. [23] argue that the power-consuming input buffers are usually placed far away from the switching core, which makes it impractical for an input scheduler to keep track of the real-time buffer occupancies at its associated crosspoints.

To avoid such implementation difficulties, Kanizo et al. [23] consider a self-sufficient single-chip crosspoint-queued (CQ) switch whose buffering and scheduling are performed solely inside the switching core, and argue for its feasibility given state-of-art ASIC technologies [25, 20, 7]. Unlike an IQ or OQ switch which may spread its buffer space on multiple input/output line-cards, the total buffer space of a single-chip CQ switch is limited by the chip size.

This may seem like a severe deficiency at first glance, since it has long been believed that Internet routers should provide one round-trip-time's equivalent of buffering to prevent link starvation. However, recent studies on high-speed Internet routers by Wischik et al. [37, 38] and McKeown et al. [1, 3] challenge this commonly used approach, and suggest that the optimal buffer size can be much smaller than that was previously believed. The reason lies in the fact that the Internet backbone links are usually driven by a large number of different flows, and multiplexing gains can be obtained under the congestion and flow control mechanisms. They also argue that short-term Internet traffic approximates the Poisson process, while long-range dependence holds in large time-scales. As a result, a much smaller amount of buffering is required as long as the traffic load is moderate, and thus can readily be accomodated on a single chip.

The single-chip CQ switch has many distinct features. On the one hand, using small segregated on-chip buffers instead of large aggregated off-chip memory allows much faster memory access on ASICs, which could have been a bottleneck for high speed switches. It also divides the scheduling and buffering tasks into small chunks, which are then fulfilled by a large number of crosspoints with low hardware requirement at each node. On the other hand, because its buffers are small and segregated, a basic CQ switch with simple scheduling algorithms, such as round-robin (RR), oldest-cell-first (OCF) and longest-queue-first (LQF), may experience far more packet drops than an IQ or OQ switch with the same total amount of buffering. Previous analyses and simulations done by Kanizo et al. [23] and Radonjic et al. [32, 34] have shown that LQF provides the highest throughput for a CQ switch in many cases, but its performance is still worse than an OQ switch with the same total buffer space. This problem is more severe when there are more ports and thus the buffer size at each crosspoint is more restricted.

A key observation here is that when a certain crosspoint experiences packet overflow, other crosspoint buffers can still be quite empty, i.e., the buffer utilizations are unbalanced. The unbalanced-utilization problem becomes worse when the incoming traffic is bursty or non-uniform. As reported in [23, 33], even LQF scheduling works poorly under these conditions. Unfortunately, analyses of real Internet traffic traces often reveal such burstiness and non-uniformity. As a result, how to efficiently use the crosspoint buffers so as to reduce packet drops remains a major issue before single-chip CQ switches can be widely accepted.

One possible solution to lessen the problem is to add an extra *load-balancing* stage in front of the original switching fabric [10, 11]. As incoming traffic passes through the first load-balancing stage, its burstiness and non-uniformity can be greatly reduced. However, the extra load-balancing stage can also introduce a mis-sequencing phenomena, i.e., packets of the same flow may not leave in the same order as they arrive. Mis-sequencing may cause unwanted performance degradation in many Internet services and applications, e.g., TCP-based data transmission. TCP remains the most dominant transport layer protocol used in the public Internet [8], but it performs poorly if the correct packet order is not maintained end-to-end, because such out-of-order packets might be treated as lost and trigger unnecessary retransmissions and congestion control [35, 4]. As a result, many network operators insist that packet ordering must be preserved in designing packet-switched Internet routers. Previous approaches to restore packet ordering include extra re-sequencing buffers [11, 26] and frame-based scheduling [14, 22], but at the cost of higher delay and buffer requirement.

Another candidate is *deflection routing*. This concept was proposed in the networking area as early as in the 1980s. The general idea is to reroute a packet to another node or path when there is no buffer available on its regular (shortest) path. Topologies proposed for deflection routing include Manhattan Street Network [29], Shuffle-Exchange Network [27], etc. All these designs effectively share distributed buffers at different nodes and lower the packet drop rate, but they also alter the packet order due to multi-path routing. Although work has been done to bound the maximum delay with deflection routing [6, 17], they have not solved the mis-sequencing problem completely.

Considering these points, we propose a chained crosspoint-queued (CCQ) switching architecture, apply load balancing and deflection routing techniques, and jointly design buffer sharing and in-order scheduling to meet the goals of low packet drop rate and correct packet order. In order to resolve the major constraints of buffering, some modifications to the basic CQ switching architecure are made, but to a modest and feasible extent. Some fast message passing and cell deflection need to be supported between adjacent crosspoints, but they can be implemented easily since the crosspoints are linked on a single chip and thus such communication is purely internal to the switch core.

We mainly consider four different configurations in this paper:

- *CQ-LQF:* This is the basic single-stage CQ switch (Section 2.1) with LQF scheduling at each output [23]. The crosspoint buffers are segregated, and no speedup is required. This serves as a benchmark (worst-case performance) to be compared with other schemes.

- *CCQ-OCF:* This refers to a two-stage CCQ switch (Section 2.2) with OCF scheduling at each output (Section 3). A load-balancing stage is added to the front, and the segregated crosspoints associated with common outputs are connected into daisy chains to support deflection routing, which requires an internal memory speedup of 2.

- *CCQ-RR:* We also propose a less-demanding RR algorithm (Section 4) to take the place of OCF scheduling in the CCQ switch. In this scheme, a *wait-counter* is tagged to each cell, aligned with other cells of the same flow through just-in-time notifications, and preserved during deflection. Upon departure, the wait-counter of a HOL cell is compared with another *RR-counter* maintained by each output scheduler to determine its eligibility of departure. In this way, the correct packet order is ensured.

- *OQ:* Finally, a typical OQ switch with the same total buffer space as the CQ switches above is considered. It can also be

viewed as a CQ switch in which crosspoints associated with the same output use a shared memory. A speedup of N is required in the worst case. This also serves as a benchmark (best-case performance) to be compared with our proposed schemes.

The rest of this paper is organized as follows. In Section 2, the basic single-chip CQ switch is briefly reviewed, and an augmented CCQ switching architecture is introduced. In Sections 3 and 4, two kinds of buffering and scheduling schemes suitable for load balancing and deflection routing are proposed and analyzed. Then we run some numerical simulations with different traffic patterns and system configurations in Section 5, verifying the effectiveness of the proposed techniques. Finally, we conclude our work in Section 6.

2. SYSTEM ARCHITECTURE

2.1 Basic Crosspoint-Queued Switch

The single-chip CQ switch [23] is a self-sufficient architecture which has all its buffers placed at the crosspoints of input and output lines. There is no buffering at input or output line-cards, as shown in Fig. 1.

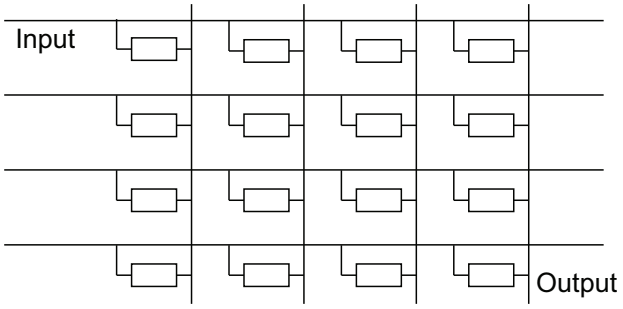


Figure 1: The single-stage basic CQ switch.

Assume an $N \times N$ CQ switch with crosspoint buffers of size B each. Then $0 \leq B(i,j) \leq B$ denotes the buffer occupancy at crosspoint (i,j), $i,j = 1,2,...,N$. We assume that the CQ switch works in a slotted manner, i.e., packets are fragmented into fixed-length cells before entering the switch core. The buffer occupancies and sizes are also measured in units of such cells. Usually a header is appended to each cell before entering the switching fabric. Such headers may contain a cell ID, source/destination ports, a time-stamp, etc.

The basic *CQ-LQF* scheduling scheme can be described as two phases in each time slot:

- ***Arrival Phase:*** For each input i, if there is a newly arriving cell destined to output j, it is directly sent to crosspoint (i,j). If buffer (i,j) is not full, i.e. $B(i,j) < B$, the new cell is accepted and buffered at the tail of line (TOL). Otherwise, this cell is dropped.

- ***Departure Phase:*** For each output j, if not all crosspoints $(*,j)$ are empty, the output scheduler picks the one with the longest queue, and serves its HOL cell.

The point of LQF rule is that it always serves the fullest buffer which is most likely to overflow. Since each output must determine the longest queue among all N crosspoints in each time slot, its worst-case time complexity is at least $O(\log N)$, assuming parallel comparator networks.

In this paper, we define that a cell belongs to flow (i,j) if it travels from input i to output j. Thus for *CQ-LQF*, cells that belong to the same flows are always served in the same order as they arrive.

2.2 Chained Crosspoint-Queued Switch

The basic CQ switch is simple and elegant. However, its buffers are small and segregated, which results in a high packet drop rate and a low buffer utilization. We therefore develop an augmented architecture, the CCQ switch, which is suitable for load balancing and deflection routing when combined with the scheduling schemes to be proposed in subsequent sections.

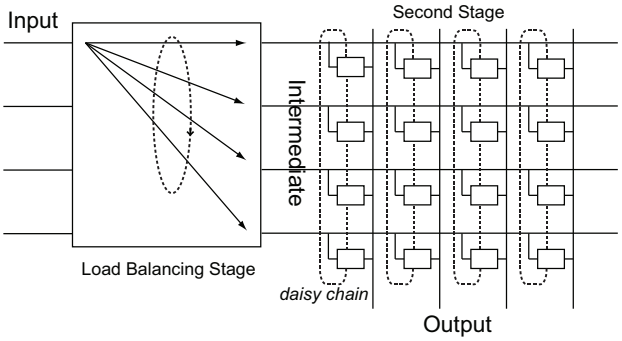


Figure 2: The two-stage CCQ switch.

In the CCQ switch, crosspoints associated with a common output port are single-connected into a daisy chain (in the order of their associated input port indices), as shown in Fig. 2. Specifically, crosspoint (i,j) is connected with its predecessor $(i-1,j)$ and successor $(i+1,j)$. Note that since the input and output port indices should always be within 1 through N, thus $(i-1,j)$ should actually be $(mod(i-2,N)+1,j)$, while $(i+1,j)$ should actually be $(mod(i,N)+1,j)$. For ease of presentation, we shall use the $(i-1,j)$ and $(i+1,j)$ notation in the rest of this paper.

With this modification, message passing and cell deflection can be easily supported between adjacent crosspoints along the daisy chains. In terms of the hardware requirement, by adding an extra layer of connections, we introduce an extra memory read/write speedup for each crosspoint buffer. The extra memory speedup and inter-crosspoint connections are purely internal to the switch core, implemented on a single chip, and thus do not impose extra burdens on the links between the input/output line-cards and the switching core (card-edge and chip-pin limitations [7]).

To further reduce the probability of buffer overflow, we also place an extra load-balancing switch (first stage) in front of the CQ switching fabric (second stage), as shown in Fig. 2. The load-balancing stage walks through a fixed sequence of configurations: at time t, it connects each input i to intermediate port $i+t$, which is also input $i+t$ of the second-stage. Note that $i+t$ is an abbreviation of $mod(i+t-1,N)+1$ for ease of presentation.

3. OLDEST-CELL-FIRST SCHEDULING FOR THE CCQ SWITCH

In [23, 32], it has been recognized that LQF provides a lower packet drop rate for the basic CQ switch than any other simple scheduling algorithms like random, RR and OCF. However, its performance can still be far worse than an OQ switch with the same total buffer space, if the incoming traffic is bursty or non-uniform.

In this section, we propose a scheme that allows different crosspoints in the same daisy chain to share packets evenly, and use the OCF scheduling algorithm to ensure correct packet ordering.

3.1 CCQ-OCF Scheduling Design

OCF is a popular scheduling algorithm which always picks the oldest cell to serve. Compared with LQF, OCF usually incurs a larger packet drop rate since it does not always serve the buffer that is most likely to overflow. Compared with RR, OCF is much more complicated since it requires repeated comparisons of time-stamps at each time slot. Despite these disadvantages, OCF is still attractive since it can easily maintain the packet order across all flows. This advantage makes OCF a good candidate to solve the mis-sequencing problem caused by load balancing and deflection routing. The performance loss due to using OCF rather than LQF can be negligible since load balancing and deflection routing already do a good job in equalizing the buffer utilizations.

In this scheme, we use the two-stage CCQ switch. Every incoming cell is assigned a time-stamp to record its arrival time. Each crosspoint needs to maintain the buffered cells in the order of non-decreasing time-stamps (i.e., first-come-first-serve). Then the output schedulers will only need to compare the time-stamps of HOL cells to determine the oldest one in each time slot.

The detailed scheme for *CCQ-OCF* is described below:

- ***Arrival Phase:***

  At time t, for each input i, if there is a newly arriving cell destined to output j, then after passing the load-balancing stage that connects input port i to intermediate port $i+t$, it is directly sent to crosspoint $(i+t, j)$ of the second stage. If the buffer is not full, i.e., $B(i+t, j) < B$, the new cell is accepted and buffered at TOL with time-stamp t. Otherwise, this overflowing cell is dropped.

- ***Departure Phase:***

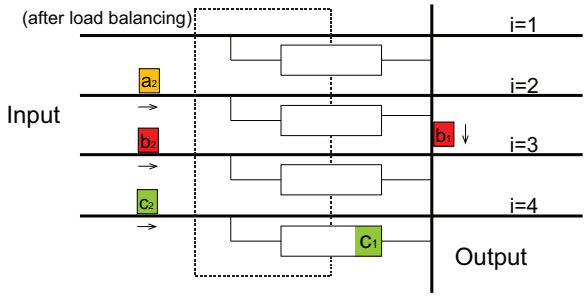
  For each output j, if there is at least one non-empty crosspoint buffer $(*, j)$, the output scheduler picks the one with the oldest HOL cell, and serves this cell.

- ***Deflection Phase:***

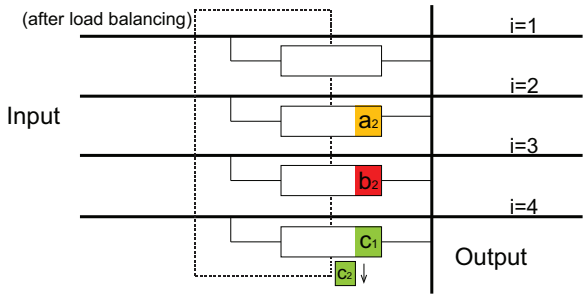
  Each crosspoint (i, j) does the following step by step:

  1. Report buffer occupancy $B(i, j)$ to its predecessor crosspoint $(i-1, j)$;
  2. Receive a buffer occupancy report $B(i+1, j)$ from its successor crosspoint $(i+1, j)$;
  3. If $B(i, j) > B(i+1, j)$, deflect the TOL cell to its successor crosspoint $(i+1, j)$;
  4. Receive a deflected cell from its predecessor crosspoint $(i-1, j)$. If there is one, insert the deflected cell into the ordered queue according to its time-stamp.

An example is illustrated in Fig. 3. Different flows are marked with different colors and alphabets, e.g., $yellow - a$, $red - b$, and $green - c$. The time-stamps are indicated by integer subscripts, e.g., $1, 2, 3$. During the departure phase, crosspoint $(2, j)$ with the oldest cell b_1 is served by output j. Then in the deflection phase, crosspoint $(4, j)$ deflects its TOL cell c_2 to its less occupied successor $(1, j)$, whereas crosspoint $(2, j)$ and $(3, j)$ do not deflect because $B(2, j) = 1 = B(3, j) < B(4, j) = 2$ at the beginning of this deflection phase. The load balancing and deflection routing mechanisms in *CCQ-OCF* aim to equalize the buffer occupancies of all crosspoints throughout the daisy chain, and thus fully utilize the limited buffer space. All cells leave the switch in non-decreasing order of time-stamps, irrespective of where they are actually buffered.



(a) Arrival and departure phases.



(b) Deflection phase.

Figure 3: An example of *CCQ-OCF*.

3.2 Features

We list some important features of *CCQ-OCF* as follows.

Property 1: There is no cell drop during deflection.

Deflections only take place when $B(i, j) > B(i+1, j)$, and the direction of deflections can only be from (i, j) to $(i+1, j)$. So $B(i+1, j)$ increases at most by 1 during the deflection phase. Since $B(i+1, j) < B(i, j) \leq B$ before deflection, it will never cause an overflow. This applies to all $i, j = 1, 2, ..., N$.

Property 2: Cells of the same flow always leave the switch in the same order as they arrive.

According to *CCQ-OCF*, incoming cells are always directly sent to the corresponding crosspoints. Cells in these crosspoints are always kept in non-decreasing order of their arrival time, even when deflections take place. Then each output scheduler picks the HOL cells according to the OCF scheduling algorithm, which strictly maintains the order of cells, even across different flows.

Insight 1: *CCQ-OCF* gains the most advantage over *CQ-LQF* in large switches with moderate crosspoint buffer size under bursty or non-uniform traffic.

First, we assume uniform Bernoulli i.i.d. traffic. Then according to the law of large numbers, the long-term buffer occupancy of each crosspoint tends to be equalized asymptotically if the buffer size is large enough, even without load balancing or deflection routing. In this case, the proposed scheme is of limited efficacy. By contrast, *CCQ-OCF* can be much more effective under bursty or non-uniform traffic, since the arrival rates at each crosspoint can be very different, especially in short time scales.

On the other hand, a large switch size boosts the multiplexing gain since the bursty traffic can be evenly distributed to more crosspoints. In terms of the crosspoint buffer size, neither should it be so large that the short-term bursty traffic is smoothed through long-

term averaging, nor should it be too small to sustain regional traffic fluctuations before deflection routing takes place to resolve them.

Insight 2: The worst-case time complexity at each crosspoint is $O(\log B)$ in each time slot, and each output scheduler needs $O(\log N)$ time to determine the oldest cell.

As mentioned before, the crosspoints need to maintain the cells in non-decreasing order of time-stamps. Observing that such ordering may only be disturbed upon cell arrival, departure and deflection, we consider the following cases:

- Newly arriving cells should always be placed at TOL since they have the largest time-stamps so far. Thus cell arrivals do not break the ordering.
- Only HOL cells can leave the switch. These cells always have the smallest time-stamps. Thus cell departures do not break the ordering either.
- Only the newest cells (TOL cells), can be deflected from highly-utilized crosspoints (sender) to their successor crosspoints (receiver). They always have the largest time-stamps at the senders. Thus cell deflections do not break the ordering at these senders.
- In each time slot, each crosspoint may receive at most one deflected cell from its predecessor. This deflected cell should then be searched and inserted into the ordered queue at the receiver according to its time-stamp.

Summing up all three phases, each crosspoint needs to perform at most $O(1)$ search, $O(1)$ insertion, and $O(1)$ deletion operations in each time slot. All these can be done in $O(\log B)$ time with a self-balancing binary search tree.

In terms of the output schedulers, they need to determine the oldest HOL cell in each time slot. One way to accomplish this is to implement a hardware-based comparator network which works like a single-elimination tournament. In each round we eliminate half of the HOL cells. Then the total amount of comparisons is $O(N)$ and the number of rounds is $O(\log N)$. An alternative way is to maintain a heap of HOL cells according to their time-stamps. In each time slot we extract the oldest HOL cell and insert some new HOL cells into the heap. A drawback of this approach is that the number of new HOL cells to be inserted can be $O(N)$ in the worst case if all crosspoints were empty in the previous time slot.

4. ROUND-ROBIN SCHEDULING FOR THE CCQ SWITCH

In the previous section, the OCF scheduling algorithm has been used to maintain the correct packet order. This method is straightforward and promising, but requires considerable computation due to repeated sorting in each time slot. On the other hand, the global packet ordering guaranteed by OCF is too strict, since we only need per-flow packet ordering. In this section, we propose a new scheme that relies on a less-demanding RR polling algorithm and an explicit notification mechanism between adjacent crosspoints to preserve per-flow packet ordering. The underlying idea is partly inspired by the Mailbox Switch [9] and Padded Frame [22], but it is implemented in a very different way here that avoids extra delays.

4.1 CCQ-RR Scheduling Design

4.1.1 Wait-Counter and RR-Counter

In this scheme, every crosspoint should maintain a *wait-counter* for each of its buffered cells, denoted by $W(i,j,k)$, $1 \le k \le B(i,j)$. Another anticipatory wait-counter for the next incoming cell, denoted by $W(i,j,B(i,j)+1)$, is also maintained by crosspoint (i,j). When a new cell arrives at (i,j), it is assigned $W(i,j,B(i,j)+1)$ upon acceptance. Then $B(i,j)$ gets incremented, and a new anticipatory wait-counter should be generated as $W(i,j,B(i,j)+1) = W(i,j,B(i,j))+1$.

As a counterpart of the wait-counters, we also let each output j maintain a ***RR-counter*** $R(j)$, in addition to its arbiter position $1 \le A(j) \le N$ which always points to the last crosspoint it has polled. $R(j)$ is incremented during each RR polling cycle when $A(j) = 1$.

Both the wait-counters $W(i,j,k)$ and the RR-counters $R(j)$ are ever-increasing (the grow-to-infinity problem can be resolved by dropping the carry when these counters exceed a sufficiently large value), but they should be maintained in non-decreasing order, so that $R(j) \le W(i,j,k) \le W(i,j,k+1)$ for any $1 \le i,j \le N$ and $1 \le k \le B(i,j)$ at any time.

An arbitrary cell k stored at a non-empty crosspoint (i,j) is eligible to leave the switch, if and only if, $W(i,j,k) = R(j)$; thus crosspoint (i,j) should refrain from being served by output j until its HOL cell becomes eligible. In terms of any empty crosspoint (i',j), it should update $W(i',j,1) = R(j)+1$ every time when output j polls it and proceeds to subsequent crosspoints.

4.1.2 Counter-Alignment Notification

We also design an explicit counter-alignment notification mechanism, which coordinates the correct packet ordering under load balancing. Such a notification is initiated by any crosspoint (i,j) upon acceptance of a newly arriving cell. It is then passed down to $(i+1,j)$ and subsequent crosspoints along the daisy chain. Upon reception, the receiver crosspoint should examine the contents, make necessary updates to its own anticipatory wait-counter, and determine whether to drop the notification message or to relay it to subsequent crosspoints.

Information contained in a notification message consists of two parts: a counter-alignment field $CA(i,j)$, which indicates the minimum wait-counter for the next incoming cell to crosspoint $(i+1,j)$, and a source-of-notification field $SN(i,j)$, which denotes the crosspoint that has initiated the message.

Specifically, when crosspoint (i,j) accepts a new cell, it immediately initiates a counter-alignment notification with $CA(i,j) = W(i,j,B(i,j))$ (increment if $i = N$) and $SN(i,j) = i$, and sends it to the successor crosspoint $(i+1,j)$ in the same daisy chain.

Then for crosspoint $(i+1,j)$, if $CA(i,j) \ge W(i+1,j,B(i+1,j)+1)$ and not $SN(i,j) = i+1$ (discard the message if it has traversed the daisy chain and come back to its origination), it updates $W(i+1,j,B(i+1,j)+1) = CA(i,j)$, and decides to relay the notification message with $CA(i+1,j) = CA(i,j)$ (increment if $i+1 = N$) and $SN(i+1,j) = SN(i,j)$ to its own successor $(i+2,j)$ in the next time slot, if by that time it has not accepted a new cell and generated a new notification message.

In this way, the mis-sequencing problem caused by load balancing can be solved. Cells of the the same flow are always assigned with non-decreasing wait-counters through just-in-time notifications between any two consecutive arrivals.

4.1.3 Deflection Routing with Counter Preserved

Deflection routing may also introduce mis-sequencing. With wait-counters, it is straightforward to resolve the issue.

Similar to *CCQ-OCF*, each crosspoint (i,j) is allowed to deflect one TOL cell to its successor $(i+1,j)$ in each time slot if $B(i,j) > B(i+1,j)$. The deflected cell should carry its own wait-counter $DW(i,j) = W(i,j,B(i,j))$ (increment if $i = N$)

with it. When crosspoint $(i+1, j)$ receives the deflected cell, it compares $DW(i, j)$ with its own cells, and inserts the deflected cell to the appropriate position to maintain non-decreasing order of wait-counters. If it has one or more cells with wait-counters equal to $DW(i, j)$, the deflected cell should be inserted in front of all of them to preserve their relative order of departure. In case $DW(i, j) \geq W(i+1, j, B(i+1, j)+1)$, update $W(i+1, j, B(i+1, j)+1) = DW(i, j)+1$.

Now that there may be multiple cells with the same wait-counters at each crosspoint (i, j), output j should adopt an exhaustive RR algorithm, serving all cells k at crosspoint (i, j) with $W(i, j, k) = R(j)$ before proceeding to the next eligible crosspoint. In this way, deflection routing will not alter the order of cells to be served.

4.1.4 CCQ-RR Scheme

- ***Arrival Phase:***

  Same as in *CCQ-OCF* except that the wait-counters should be assigned and updated according to Section 4.1.1 instead of the time-stamps.

- ***Notification Phase:***

  Each crosspoint (i, j) sends and receives a counter-alignment notification message according to Section 4.1.2.

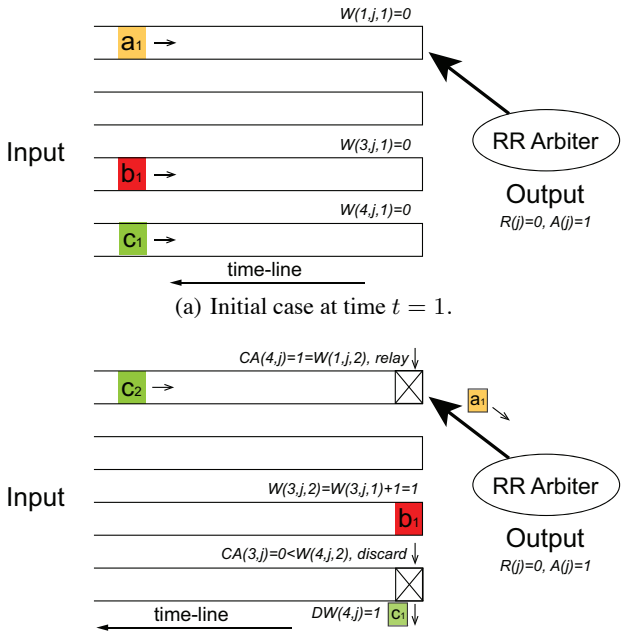
- ***Departure Phase:***

  Each output j polls its associated crosspoints $(*, j)$ in an exhaustive RR fashion, starting from its final position $A(j)$ in the previous time slot. The polling process continues until output j serves an eligible crosspoint with $W(i, j, 1) = R(j)$, or it finds all buffers empty.

- ***Deflection Phase:***

  Same as in *CCQ-OCF* except that wait-counters take the place of time-stamps according to Section 4.1.3.

An example is illustrated in Fig. 4. Different flows are marked with different colors and alphabets, e.g., $yellow - a$. The time-stamps (for illustration, not required in implementation) are indicated by integer subscripts, e.g., 1,2,3. Wait-counters are represented by their positions on the time-line, while vacancies (cross-marked squares) in the time-lines do not occupy real buffer positions. During the arrival phase at time $t = 1$, the new cell b_1 is tagged with wait-counter $W(3, j, 1) = 0$, and $W(3, j, 2) = W(3, j, 1)+1 = 1$ is generated. Similar tagging and updates occur for cells a_1 and c_1. Next, during the notification phase, crosspoint $(3, j)$ initiates a counter-alignment notification with $CA(3, j) = W(3, j, 1) = 0$ for the newly accepted cell b_3, and sends it to its successor $(4, j)$, but this message is discarded because $CA(3, j) = 0 < W(4, j, 2) = 1$. On the other hand, crosspoint $(4, j)$ also initiates a counter-alignment notification $CA(4, j) = W(4, j, 1) + 1 = 1$ (note that $i = 4 = N$ here) for c_1. Crosspoint $(1, j)$ accepts the notification, updates $W(1, j, 2) = CA(4, j) = 1$ so that the next incoming cell c_2 will be served later than c_1, and decides to relay this message to subsequent crosspoints in future time slots. Then during the departure phase, the first eligible cell a_1 with $W(1, j, 1) = 0 = R(j)$ is served by the output, leaving a vacancy in the time-line. Finally, during the deflection phase, crosspoint $(4, j)$ finds its successor $(1, j)$ less occupied, so it deflects the TOL cell c_1 with $DW(4, j) = W(4, j, 1) + 1 = 1$ (again, $i = 4 = N$). As a result, the new cell c_2 to arrive at time $t = 2$ will be pushed back to the $3rd$ time-line position, although it is stored in the $2nd$ buffer position. The cells shall leave the switch in the order of a_1, b_1, c_1, c_2, etc.



(a) Initial case at time $t = 1$.

(b) Changes until time $t = 2$.

Figure 4: An example of *CCQ-RR*.

4.2 Features

Some key features of *CCQ-RR* are listed below.

Property 1: The proposed *CCQ-RR* scheme is work-conserving, if the maximum number of deflections is restricted to K, and each output can perform $N + K + 1$ polls in each time slot.

First, consider the situation without deflection routing. Pick any arbitrary cell $\mathcal{X}_1$ that arrives at crosspoint (i, j) and gets wait-counter $W(i, j, k)$.

- If $W(i, j, k)$ was updated upon acceptance of a newly arriving or deflected cell $\mathcal{Y}$, then $\mathcal{Y}$ must have been exactly $N+1$ polls away at that time, and may have become even closer after deflections. If $\mathcal{Y}$ is served immediately before $\mathcal{X}$, then the output arbiter needs at most $N+1$ polls to reach $\mathcal{X}$. Otherwise, if any other cell $\mathcal{Z}$ is the immediate predecessor (in the order of departure), $\mathcal{Z}$ must be at most $N+1$ polls away.

- If $W(i, j, k)$ was updated through counter-alignment initiated for cell $\mathcal{Y}$, then $\mathcal{Y}$ must have been at most N polls away at that time, otherwise the counter-alignment notification should have already been discarded after traversing the daisy chain. If $\mathcal{Y}$ is served immediately before $\mathcal{X}$, then the output arbiter needs at most N polls to reach $\mathcal{X}$. Otherwise, if any other cell $\mathcal{Z}$ is the immediate predecessor (in the order of departure), $\mathcal{Z}$ must be at most N polls away from $\mathcal{X}$.

- Otherwise, $W(i, j, k)$ must have been updated when crosspoint (i, j) was empty through $W(i, j, 1) = R(j) + 1$, then $k = 1$ and it is at most N polls away from the output arbiter.

Summing up all three conditions, the output arbiter needs at most $N + 1$ polls (starting from its last polled crosspoint) in each time slot to ensure it is work-conserving.

We next take deflection routing into account. If the number of deflections is limited to K, then the gap between any two consecutive cells (in the order of departure) is enlarged by at most K polls. As a result, each output arbiter needs at most $N+1+K$ polls in each time slot to be work-conserving.

Property 2: Cells of the same flow always leave the switch in the same order as they arrive.

For load balancing, cell order is preserved through just-in-time counter-alignment notifications between any two consecutive arrivals of the same flow. In terms of deflection routing, it will not alter the order of departure if the wait-counters are preserved and adjusted when necessary. These are elaborated in Sections 4.1.2 and 4.1.3, and some boundary conditions should be taken care of. Specifically, the last crosspoint (N, j) in each daisy chain j must always increment the counter-alignment field $CA(N, j)$, as well as the wait-counter of its deflected cell $DW(N, j)$, so as to match with the starting point of a new RR polling cycle.

Insight 1: *CCQ-RR* gains the most advantage over *CQ-LQF* in large switches under bursty or non-uniform traffic.

The reasons are similar as for *CCQ-OCF* and thus omitted.

Insight 2: The worst-case time complexity at each crosspoint is $O(\log B)$ in each time slot, and each output scheduler can find the next eligible HOL cell in $O(\log N)$ time.

The operations at each crosspoint in *CCQ-RR* are exactly the same as those in *CCQ-OCF*, except that the time-stamps are replaced with the wait-counters, and that $O(1)$ additional updates to the anticipatory wait-counters need to performed. All these can still be accomplished in $O(\log B)$ time using a self-balancing binary search tree.

In terms of the output scheduler, each RR arbiter may find the next eligible crosspoint within $O(\log N)$ time using a hardware-based priority encoder [18] (typically a few nanoseconds). Although the magnitude of time complexity for RR looks the same as that for OCF, the constant factor can be much smaller, and it has been widely recognized that RR is much easier to implement than OCF. On the other hand, in order to utilize the priority encoder, each output arbiter j may need to broadcast its RR-counter $R(j)$ and arbiter position $A(j)$, so that each crosspoint may determine its own eligibility in a distributed manner.

5. NUMERICAL SIMULATIONS

In this section, we perform numerical simulations with MATLAB to show the performance improvements through load balancing and deflection routing. Specifically, we compare the cell drop rates and critical buffer utilizations of the CCQ switches against a basic LQF-based CQ switch and an OQ switch with the same total buffer space. The latter two systems are used as benchmarks in our comparison.

The *cell drop rate* is the average probability a random cell is dropped by the switch. We shall focus on the drop rate of fixed-length cells after fragmentation. Cell drop rate should be as low as possible, but we set up a reasonable target at 10^{-5} for the following reasons:

- The state-of-art Internet end-to-end loss rate for IP packets is in the order of 10^{-3} to 10^{-2} [5, 21];
- Empirical results reveal that TCP/IP protocols may tolerate an end-to-end loss rate of 10^{-3} and still yield satisfactory performances [19];
- Measurements on the Internet show that the average end-to-end hop-count is of the order of tens [2, 12];
- Assume that the variable-length IP packets are fragmented into $64byte$ cells, then the average number of segments for each IP packet is of the order of tens [8].

The *critical buffer utilization* is defined as the average utilization of all buffers $(*, j)$ when a cell destined to output j is dropped, i.e., $\eta_{cq} = E(\frac{\sum_{i=1}^{N} B(i,j)}{N \times B} |$ cell drop at daisy chain $j)$ for a CQ switch, and $\eta_{oq} \equiv 100\%$ for an OQ switch. We shall see later that the critical buffer utilization is negatively correlated with the cell drop rate.

In the rest of this paper, we shall investigate the impact of traffic load, non-uniformity, burstiness and switch size on *CQ-LQF*, *CCQ-OCF*, *CCQ-RR* and *OQ* schemes using various synthesized traffic patterns and real Internet traces.

5.1 Impact of Traffic Load

First, we evaluate the effectiveness of the proposed schemes under uniform bursty traffic. The destinations of incoming cells are evenly distributed among all N output ports, i.e., $\lambda_{ij} = \frac{\lambda}{N}, i, j = 1, 2, ..., N$, where $0 \leq \lambda \leq 1$ is the normalized traffic load.

Since real Internet traffic is usually bursty and long-range dependent (LRD), we shall focus on this kind of traffic. Specifically, we use the Markov Chain model in [16] to generate LRD traffic with Hurst parameter $H = 0.75$ and maximum length $L = 1000$, i.e., each single burst of cells belonging to the same flow may last for at most 1000 time slots. Subsequently, we shall use this traffic-generating model, and adjust H, L, λ_{ij} to control the traffic pattern.

We consider 32×32 switches with crosspoint buffer size $B = 40$ cells. The simulation lasts $T = 10^7$ time slots.

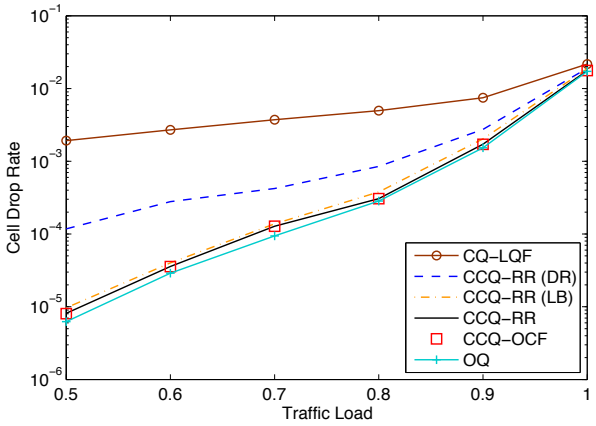


Figure 5: Cell drop rate of 32×32 switches with $B = 40$ under uniform bursty traffic with $H = 0.75$, $L = 1000$ and $0.5 \leq \lambda \leq 1.0$.

Fig. 5 compares the cell drop rates of various schemes. The abbreviation "*CCQ-RR (LB)*" stands for "*CCQ-RR* with load balancing only", and "*CCQ-RR (DR)*" means "*CCQ-RR* with deflection routing only". These two degenerate versions of *CCQ-RR* are compared here so as to demonstrate the respective effectivenesses of load balancing and deflection routing. They also preserve the correct packet order.

Simulation results show that *CCQ-OCF* and *CCQ-RR* have the lowest cell drop rates, which are much better than that of *CQ-LQF* and very close to that of *OQ*, going down to about 10^{-5} when the traffic load is $\lambda = 0.5$. Similar performances can also be achieved under higher traffic loads if larger buffers are implemented.

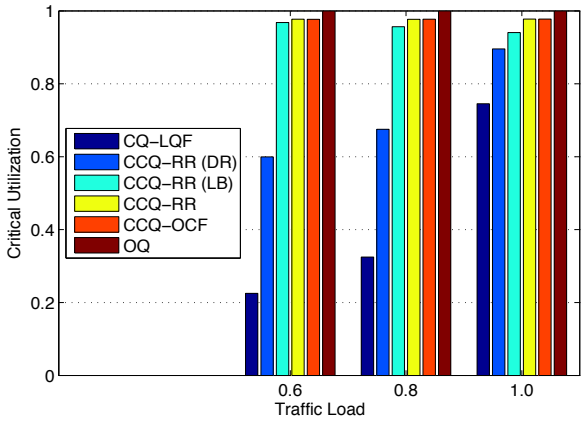


Figure 6: Critical buffer utilization of 32×32 switches with $B = 40$ under uniform bursty traffic with $H = 0.75$, $L = 1000$ and $0.5 \leq \lambda \leq 1.0$.

Comparing *CCQ-RR (LB)* with *CCQ-RR (DR)*, we can find that deflection routing does not contribute as much as load balancing in this case. However, one cannot conclude that deflection routing is ineffective if load balancing is employed. In fact, the superiority of load balancing could largely be attributed to how we model the LRD traffic. As mentioned before, our model generates separate bursts of cells that belong to different flows, which makes load balancing especially effective. On the other hand, in real Internet traffic, such bursts should be interleaved, showing Poisson characteristics in short time scales, and leaving more time for deflection routing to propagate. Besides, load balancing is a passive mechanism, while deflection routing is a reactive strategy and its advantage can be very significant under adversarial traffic patterns.

We also compare the buffer utilizations of different schemes in Fig. 6. Here we can see that the critical utilization of *CQ-LQF* is fair when the traffic load is high (about 70% when $\lambda = 1.0$), but drops quickly as the traffic load becomes lower (only 20% when $\lambda = 0.6$). To understand this, we must realize that a lower traffic load does not necessarily lead to less burstiness according to our model, since the Hurst parameter does not change at all. Ironically, when the traffic load is lower, the incoming traffic at different cross-points can be even more unbalanced in a short time-scale. This kind of low buffer utilization leads to a larger performance degradation when the traffic load is low (as compared with *OQ*). By contrast, *CCQ-OCF* and *CCQ-RR* are not affected by the change of traffic load, showing robustness against various traffic loads.

Comparing Fig. 5 and Fig. 6, we can see a clear trend that the cell drop rate is negatively correlated with the critical buffer utilization given the same incoming traffic. The critical buffer utilizations of *CCQ-OCF* and *CCQ-RR* are close to 100%, which is only achievable by the OQ switch. Thus the significant performance improvements of the proposed schemes can be attributed to their efficient buffer sharing mechanisms, i.e., load balancing and deflection routing.

5.2 Impact of Non-uniformity

In addition to the uniform bursty traffic, we also test the proposed buffering and scheduling techniques under non-uniform traffic. In this case, the destinations of incoming cells are not evenly distributed among all N outputs. Instead, we adopt a hot-spot traffic model as follows:

$$\lambda_{ij} = \begin{cases} \rho\lambda & \text{if } i = j \\ \frac{(1-\rho)\lambda}{N-1} & \text{otherwise} \end{cases}$$

We still focus on 32×32 CQ switches with buffer size $B = 40$ cells. The incoming traffic is LRD with $H = 0.75$ and $L = 1000$, and the hot-spot parameter is set to $\rho = 0.5$.

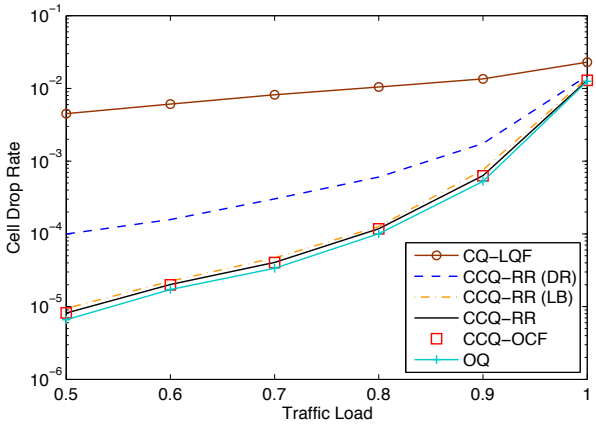


Figure 7: Cell drop rate of 32×32 switches with $B = 40$ under non-uniform bursty traffic with $H = 0.75$, $L = 1000$ and $0.5 \leq \lambda \leq 1.0$.

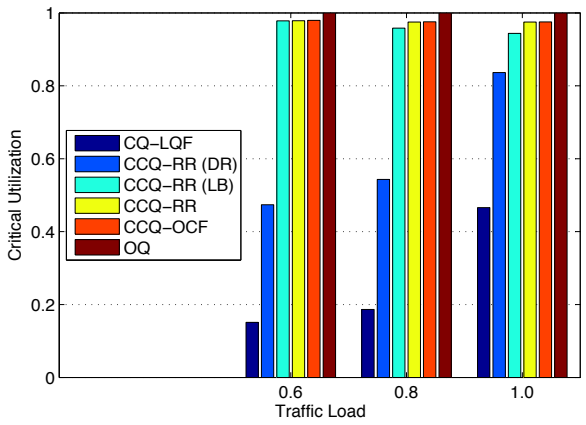


Figure 8: Critical buffer utilization of 32×32 switches with $B = 40$ under non-uniform bursty traffic with $H = 0.75$, $L = 1000$ and $0.5 \leq \lambda \leq 1.0$.

The cell drop rates and critical buffer utilizations of the proposed schemes under hot-spot LRD traffic are illustrated in Fig. 7 and Fig. 8 respectively. Comparing these two figures with their counterparts under uniform bursty traffic in Section 5.1, we find that *CQ-LQF* performs worse under non-uniform traffic, as indicated by a higher cell drop rate and a lower critical buffer utilization. By contrast, *CCQ-OCF* and *CCQ-RR* have slightly lower cell drop rates and higher critical buffer utilizations under non-uniform bursty traffic, demonstrating the same trend as *OQ*. These results show that the proposed schemes are relatively better under non-uniform traffic. We also notice that deflection routing suffers more from the non-uniformity of the incoming traffic, e.g., the critical buffer utilization

drops from 60% under uniform bursty traffic with $\lambda = 0.6$ to below 50% in this case.

5.3 Impact of Burstiness

The impact of burstiness on the performance of different schemes is also investigated. Here we set the crosspoint buffer size to $B = 40$, fix the maximum length to $L = 1000$, then change the Hurst parameter $0.6 \leq H \leq 0.9$.

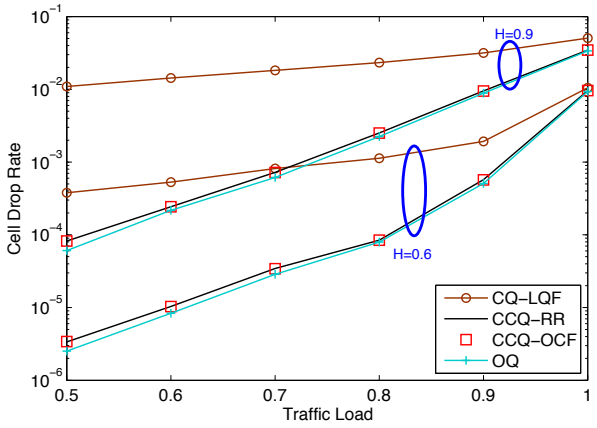


Figure 9: Cell drop rate of 32×32 switches with $B = 40$ under uniform bursty traffic with $0.6 \leq H \leq 0.9$, $L = 1000$ and $0.5 \leq \lambda \leq 1.0$.

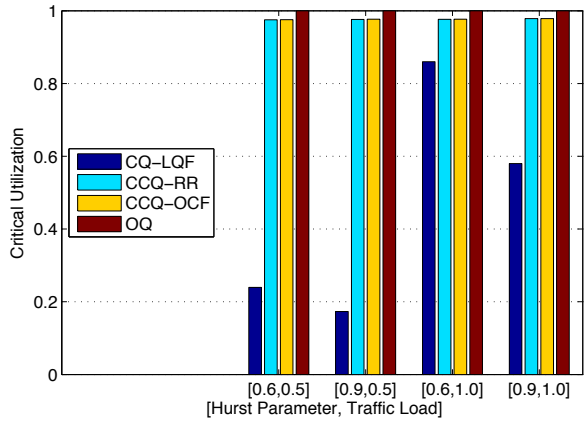


Figure 10: Critical buffer utilization of 32×32 switches with $B = 40$ under uniform bursty traffic with $0.6 \leq H \leq 0.9$, $L = 1000$ and $0.5 \leq \lambda \leq 1.0$.

Simulation results in Fig. 9 and Fig. 10 show that *CQ-LQF* performs worse when the traffic is more bursty but lower loaded. On the other hand, the proposed *CCQ-OCF* and *CCQ-RR* schemes are not affected much (as compared with *OQ*), demonstrating their robustness against different burstiness levels. The underlying reason is that the small crosspoint buffers become less capable to sustain the traffic fluctuations as the incoming cells become more bursty and intermittent, and depend more on load balancing and deflection routing to smooth the traffic. As a conclusion, the proposed schemes gain relatively larger advantages under highly bursty and intermittent traffic.

5.4 Impact of Large Switch Size

Till now, we have examined the performances of 32×32 switches under various traffic patterns. What if the switch becomes larger, i.e., with more input and output ports? Here we consider a large 128×128 CQ switch, and investigate the impact of large N on different switch configurations.

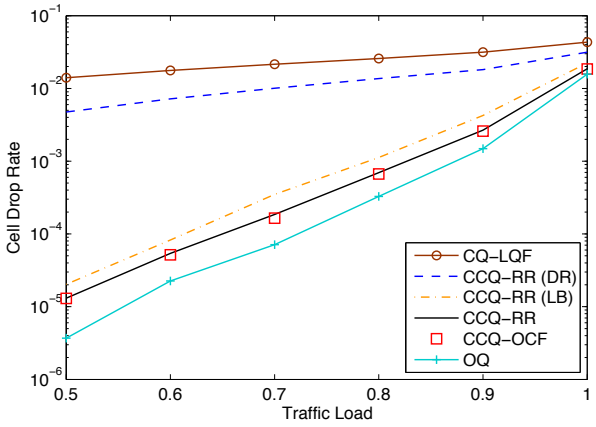


Figure 11: Cell drop rate of 128×128 switches with $B = 10$ under uniform bursty traffic with $H = 0.75$, $L = 1000$ and $0.5 \leq \lambda \leq 1.0$.

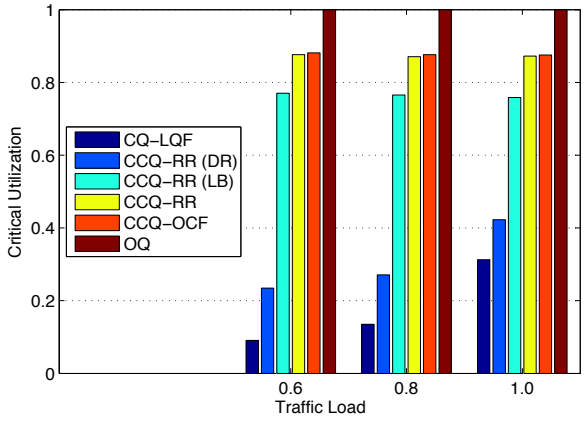


Figure 12: Critical buffer utilization of 128×128 switches with $B = 10$ under uniform bursty traffic with $H = 0.75$, $L = 1000$ and $0.5 \leq \lambda \leq 1.0$.

From Fig. 11 and Fig. 12, we can see that the legacy *CQ-LQF* method suffers from a higher cell drop rate due to a smaller crosspoint buffer size. *CCQ-OCF* and *CCQ-RR* gain a larger advantage over *CQ-LQF* in this case, but are inferior to *OQ* due to increased difficulties in buffer-sharing along longer daisy chains of smaller crosspoint buffers. Notwithstanding this issue, we may still infer that the proposed schemes are more suitable for large switches with small crosspoint buffers. We also notice that deflection routing becomes much less effective when N grows larger, because its buffer-sharing effect is local and requires more time to propagate (due to the constraint of $B(i,j) \geq B(i+1,j)+1$) than load balancing.

A larger switch size of $N = 128$ needs additional buffer space to achieve the same satisfactory cell drop rates as before. For

CQ-LQF, the total buffer space required to achieve similar performances may scale as $\Theta(N^2)$, since each crosspoint buffer should at least tolerate a single burst, whose length does not shrink much as N increases. By contrast, for *CCQ-OCF*, *CCQ-RR* and *OQ*, the total buffer space required to achieve similar performances does not scale so poorly. Compared with the case in Section 5.1, even though the switch size is 4 times larger than before, the aggregated buffer size for each output does not change at all, i.e., $N \times B = 128 \times 10 = 32 \times 40 = 1280 cells$, and the total buffer space of all outputs scales as $\Theta(N)$.

For an OQ switch, this is easy to understand, since the traffic load at each output always equals to $0.5 \leq \lambda \leq 1$, and does not change with different switch sizes. If we assume Poisson arrival processes at each input, the output queue length distributions are always the same, irrespective of N. The LRD arrival process is certainly different, but as long as the burst length is not too large compared with the output buffer size, the performance of *OQ* stays approximately the same. *CCQ-OCF* and *CCQ-RR* may also share the segregated crosspoint buffers efficiently. That is why the total amount of buffers in each daisy chain stays almost the same for a given traffic level and loss performance.

5.5 Real Internet Traces

Finally, we test the proposed schemes using real Internet traces. In the simulation, a different CAIDA OC-192 ($10 Gbps$) trace [15] is fed into each input port of the CQ switch. The incoming packets are hashed according to a fixed look-up table, so that the outputs work at approximately the same load. Variable-length IP packets are fragmented into fixed-length cells of $64 byte$ each, which is a common value used in Internet core switches.

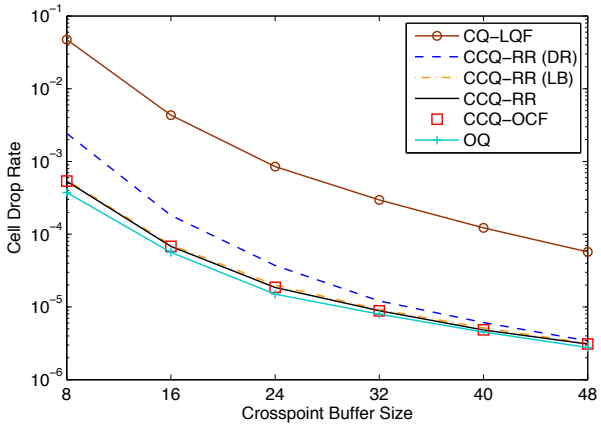


Figure 13: Cell drop rate of 32×32 switches with $8 \leq B \leq 48$ under real Internet traces with $\lambda \approx 0.45$ and $H \approx 0.75$.

First we consider a 32×32 CQ switch, and use the original traces from CAIDA with an average traffic load of $\lambda \approx 0.45$ and a measured Hurst parameter of $H \approx 0.75$. The simulation period is $T = 10^7$ time slots. Examination of the packet headers reveals that over $50,000$ flows with different source/destination IP addresses are multiplexed into each link during the simulation period. As displayed in Fig. 13, *CCQ-OCF* and *CCQ-RR* ensure very low cell drop rates, about 10 to 100 times lower than the basic LQF-based CQ switch, and close to the OQ switch with the same total buffer space. To support an average cell drop rate of 10^{-5}, only about $32 \times 32 \times 40 \times 64 byte = 2.5 Mbyte$ total buffer space is needed, thus even larger Internet core switches can be accommodated onto

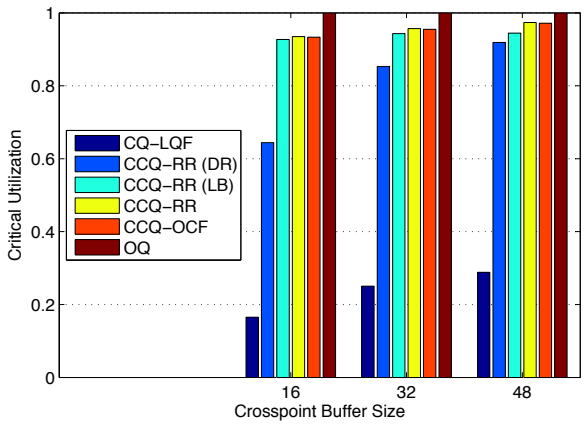


Figure 14: Critical buffer utilization of 32×32 switches with $8 \leq B \leq 48$ under real Internet traces with $\lambda \approx 0.45$ and $H \approx 0.75$.

a single chip. Also note that deflection routing contributes more as the crosspoint buffer size grows larger, and becomes almost as effective as load balancing when $B = 48$.

Comparison of the critical buffer utilizations show that all schemes achieve higher buffer utilizations with larger crosspoint buffers. However, *CCQ-OCF* and *CCQ-RR* can achieve about 90% buffer utilization when the buffer size is as small as $B = 8$, whereas *CQ-LQF* requires a much larger buffer size to smooth the traffic. Deflection routing also requires a modest buffer size to achieve high utilization, as predicted in Section 3.2, with a boost in the buffer utilizations after load balancing is applied as well. The advantages of the proposed schemes are clearly demonstrated.

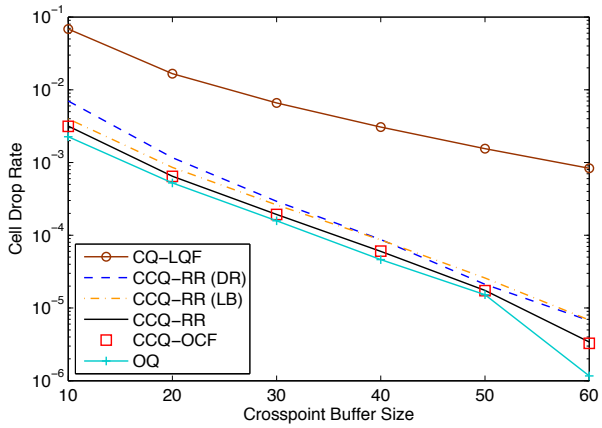


Figure 15: Cell drop rate of 128×128 switches with $10 \leq B \leq 60$ under real Internet traces with $\lambda = 0.7$.

We then consider a larger 128×128 CQ switch. We use the same Internet traces (with a different look-up table), but reduce the core switching speed and place throttles right before the input ports so that the system effectively works at a higher traffic load of $\lambda = 0.7$. The cell drop rates and buffer utilizations are shown in Fig. 15 and Fig. 16 respectively. In this case, a much larger buffer space, $128 \times 128 \times 60 \times 64 byte = 60 Mbyte$, is required to achieve the same cell drop rate of under 10^{-5}, but it is still feasible using state-of-art ASIC technologies [25, 20, 7]. The relative

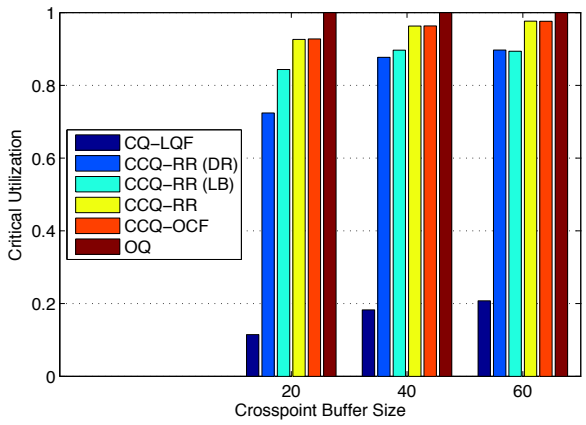


Figure 16: Critical buffer utilization of 128×128 switches with $10 \leq B \leq 60$ under real Internet traces with $\lambda = 0.7$.

performance gains of the proposed schemes over *CQ-LQF* are even higher in this case. Also note that the deflection routing mechanism in *CCQ-RR* works better than load balancing in this case, showing the robustness of such a reactive strategy against varying burstiness and non-uniformity.

6. CONCLUSION

In this paper, we address the crucial buffering constraints in a single-chip CQ switch. At the cost of some modest hardware modifications and memory speedup, we make it possible for the segregated buffers at different crosspoints to be dynamically shared along daisy chains, effectively mimicking an OQ switch. At the same time, the proposed scheduling schemes can also maintain the correct packet ordering with low complexity, which is also important in designing packet-switched networks. Exploiting the benefits of load balancing and deflection routing, we significantly improve the buffer utilization and reduce the packet drop rate, especially for large switches with small crosspoint buffers under bursty and non-uniform traffic. Extensive simulations have been performed to demonstrate that the memory sizes available using current ASIC technology is sufficient to deliver a satisfactory packet loss performance with a single-chip CQ architecture.

As part of the future work, we may improve the scheduling algorithms, and also mathematically evaluate or bound their performances. In addition, we will explore other efficient buffering techniques, and push the limits of buffer sharing across different outputs to achieve higher multiplexing gains. Other extensions like the support of QoS for packets with different priorities and multicasting are also worthy of investigation.

7. ACKNOWLEDGEMENT

This work is supported by the New York State Center for Advanced Technology in Telecommunications (CATT) and the Wireless Internet Center for Advanced Technology (WICAT) at Polytechnic Institute of New York University, Brooklyn, NY, USA.

8. REFERENCES

[1] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proc. ACM SIGCOMM*, pages 281–292, 2004.

[2] F. Begtasevic and P. V. Mieghen. Measurements of the hopcount in Internet. In *Passive and Active Measurements*, pages 183–190, 2001.

[3] N. Beheshti, Y. Ganjali, R. Rajaduray, D. Blumenthal, and N. McKeown. Buffer sizing in all-optical packet switches. In *Proc. Optical Fiber Communication Conference (OFC)*, Mar. 2006.

[4] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM SIGCOMM Comput. Commun. Rev.*, 32:20–30, 2002.

[5] M. Borella, D. Swider, S. Uludag, and G. Brewster. Internet packet loss: measurement and implications for end-to-end Qos. In *Proc. ICPP*, pages 3–12, 1998.

[6] J. Brassil and R. Cruz. Bounds on maximum delay in networks with deflection routing. *IEEE Trans. Parallel Distrib. Syst.*, 6(7):724–732, 1995.

[7] C. Minkenberg and R.P. Luijten and F. Abel and W. Denzel and M. Gusat. Current issues in packet switch design. *ACM SIGCOMM Comput. Commun. Rev.*, 33(1):119–124, 2003.

[8] CAIDA. CAIDA Internet Data - Realtime Monitors. Online: `http://www.caida.org/data/realtime`.

[9] C. Chang, D. Lee, and Y. J. Shih. Mailbox switch: a scalable two-stage switch architecture for confict resolution of ordered packets. In *Proc. IEEE INFOCOM*, volume 3, pages 1995–2006, 2004.

[10] C.-S. Chang, D.-S. Lee, and Y.-S. Jou. Load balanced Birkhoff-von Neumann switches, Part I: one-stage buffering. *Comput. Commun.*, 25:611–622, 2002.

[11] C.-S. Chang, D.-S. Lee, and C.-M. Lien. Load balanced Birkhoff-von Neumann switches, Part II: multi-stage buffering. *Comput. Commun.*, 25:623–634, 2002.

[12] X. Chen, L. Xing, and Q. Ma. A distributed measurement method and analysis on Internet hop counts. In *Proc. ICCSNT*, volume 3, pages 1732–1735, 2011.

[13] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input/output-queued switch. *IEEE J. Sel. Areas Commun.*, 17(6):1030–1039, 1999.

[14] S.-T. Chuang, S. Iyer, and N. McKeown. Practical algorithms for performance guarantees in buffered crossbars. In *Proc. IEEE INFOCOM*, volume 2, pages 981–991, 2005.

[15] K. Claffy, D. Anderson, and P. Hick. The CAIDA Anonymized 2011 IPv6 Day Internet Traces. Online: `http://www.caida.org/data/passive/passive_2011_ipv6day_dataset.xml`.

[16] R. G. Clegg and M. Dodson. Markov chain-based method for generating long-range dependence. *Phys. Rev. E*, 72(2), Aug. 2005.

[17] W. Dobosiewicz and P. Gburzynski. A bounded-hop-count deflection scheme for Manhattan-street networks. In *Proc. IEEE INFOCOM*, volume 1, pages 172–179, 1996.

[18] P. Gupta and N. McKeown. Designing and implementing a fast crossbar scheduler. *IEEE Micro*, 19(1):20–28, 1999.

[19] D. Hayes and G. Armitage. Improved coexistence and loss tolerance for delay based TCP congestion control. In *Proc. IEEE LCN*, pages 24–31, 2010.

[20] International Technology Roadmap for Semiconductors (ITRS). Executive summary. 2011.

[21] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. *IEEE/ACM Trans. Networking*, 15(1):54–66, 2007.

[22] J. Jaramillo, F. Milan, and R. Srikant. Padded frames: a novel algorithm for stable scheduling in load-balanced switches. *IEEE/ACM Trans. Networking*, 16(5):1212–1225, 2008.

[23] Y. Kanizo, D. Hay, and I. Keslassy. The crosspoint-queued switch. In *Proc. IEEE INFOCOM*, pages 729–737, 2009.

[24] M. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Trans. Commun.*, 35(12):1347–1356, 1987.

[25] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos. Variable packet size buffered crossbar (CICQ) switches. In *Proc. IEEE ICC*, volume 2, pages 1090–1096, 2004.

[26] I. Keslassy and N. McKeown. Maintaining packet order in two-stage switches. In *Proc. IEEE INFOCOM*, volume 2, pages 1032–1041, 2002.

[27] D. Lawrie and D. Padua. Analysis of message switching with shuffle/exchange in multiprocessors. In *Proc. Workshop on Interconnection Networks*, pages 116–123, 1980.

[28] Y. Li, S. Panwar, and H. J. Chao. On the performance of a dual round-robin switch. In *Proc. IEEE INFOCOM*, volume 3, pages 1688–1697, 2001.

[29] N. Maxemchuk. Routing in the Manhattan street network. *IEEE Trans. Commun.*, 35(5):503–512, 1987.

[30] N. Mckeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Networking*, 7:188–201, 1999.

[31] M. Nabeshima. Performance evaluation of a combined input-and-crosspoint-queued switch. *IEICE Trans. Commun.*, E83-B(3):737–741, 2000.

[32] M. Radonjic and I. Radusinovic. Average latency and loss probability analysis of crosspoint queued crossbar switches. In *Proc. ELMAR*, pages 203–206, 2010.

[33] M. Radonjic and I. Radusinovic. Buffer length impact to crosspoint queued crossbar switch performance. In *Proc. IEEE MELECON*, pages 119–124, 2010.

[34] M. Radonjic and I. Radusinovic. Impact of scheduling algorithms on performance of crosspoint-queued switch. *Ann. Telecommun.*, 66(5-6):363–376, 2011.

[35] S. Rewaskar. *Real world evaluation of techniques for mitigating the impact of packet losses on TCP performance*. PhD thesis, Univ. North Carolina, Chapel Hill, 2008.

[36] L. Tassiullas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multi-hop radio networks. *IEEE Trans. Autom. Control*, 37(12):1936–1949, 1992.

[37] D. Wischik. Buffer requirements for high-speed routers. In *Proc. ECOC*, volume 5, pages 23–26, 2005.

[38] D. Wischik and N. McKeown. Part I: buffer sizes for core routers. *ACM SIGCOMM Comput. Commun. Rev.*, 35(3):75–78, 2005.

[39] S. Ye, Y. Shen, and S. Panwar. DISQUO: A distributed 100% throughput algorithm for a buffered crossbar switch. In *Proc. IEEE Workshop on HPSR*, 2010.