Load Balancing in a Switch Without Buffers

Saad Mneimneh, saad@alum.mit.edu, Visiting Professor, Hunter College of CUNY, New York, NY 10021, USA

***Abstract*— A load balanced switch consists of k output queued switches, each running at a speedup (the ratio of internal memory speed to external line speed) of N/k, where N is the number of input and output ports. With such an architecture, and for some classes of traffic patterns, simply spreading the traffic among the k switches achieves 100% throughput (thus the term load balancing). However, reordering among packets of the same flow may occur, which becomes a major concern. Input and output buffers have been used in the literature to avoid this problem. This paper proves two main results: (1) it is impossible to achieve 100% throughput and no reordering for a load balanced switch without buffers, and (2) it is possible to achieve 100% throughput and limited reordering for a load balanced switch without buffers. The latter result implies that it is possible to achieve 100% throughput and no reordering for a load balanced switch with output buffers only.**

***Index Terms*—Load balancing, switching, reordering, throughput.**

I. Introduction

LOAD balancing has recently become an important aspect of a switch architecture due to its capability of spreading the traffic among multiple switching components [19], and overcoming the traditional limitations of output queued switches and input queued switches. We briefly review these limitations below:

A. Output queued switch

An output queued switch simply consists of N FIFO queues, where N is the number of output ports (without loss of generality N is also the number of input ports). The operation of the output queued switch is very trivial: when a packet destined to output j arrives at a given input, it is placed in the j^{th} queue, hence the term output queued. A simple schematic of an output queued switch is depicted in Figure 1.

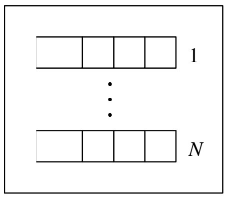

Fig. 1. Output queued switch with FIFO queues

Although idealistic, this switch suffers from a major scalability issue: up to N packets (one per input) may be placed in the j^{th} queue in one time step; hence a speedup (the ratio of internal memory speed to external line speed) of N is required.

An extended paper on load balancing algorithms in a switch and three properties: throughput, reordering, and starvation, is in preparation and will be published soon.

B. Input queued switch

To avoid sending multiple packets to the same FIFO queue (and hence the speedup of N), the input queued switch gives each input its own local copy of the switch of Figure 1, hence the term input queued switch. Figure 2 illustrates this architecture.

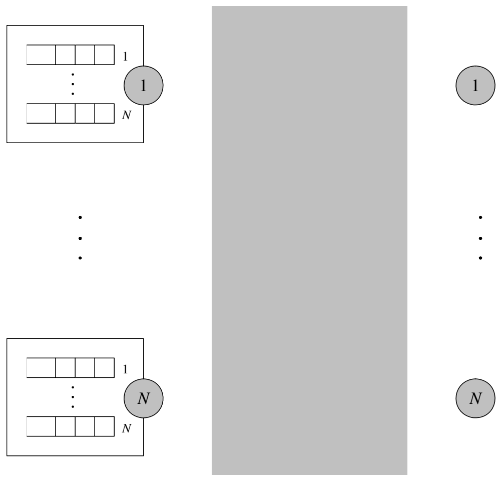

Fig. 2. Input queued switch

We now refer to the j^{th} FIFO queue at input i as VOQ_{ij} (which stands for Virtual Output Queue). VOQ_{ij} holds therefore packets at input i destined to output j.

Since at most one packet per input can arrive in every time step, all VOQs may now run at no speedup. However, every output must serve at most one input (no queues at the output side). This, with some additional implementation detail, has traditionally imposed a non-trivial operation by which a *matching* among input and output ports must be computed in every time step (see for instance [13], [12], [14], [15], [5], [8], [4], [11], and [18] for literature on this switch). A packet in VOQ_{ij} is sent across the switch only if input i is matched to output j. The matching algorithm completely determines the schedule by which packets leave their input ports, and hence makes it difficult to provide guarantees similar to an output queued switch.

C. Load balanced switch

The load balanced switch was suggested repeatedly in literature over the past few years, e.g. [16], [7], [1], and [10]. In particular, the work in [10] provides a comprehensive overview of the load balanced switch and a number of useful contributions. It is the work of [1] that first identified and emphasized the load balancing aspect of such architecture.

While the input queued switch replicates the output queued switch of Figure 1 at every input port to completely eliminate speedup (see Figure 2), a load balanced switch uses a number of parallel copies, say k, with limited speedup each. Unlike the input queued switch, however, each of the k output queued switches is accessible by all input ports (and all output ports). Figure 3 depicts the architecture of a load balanced switch. Again, we refer to the j^{th} FIFO queue in switch l as VOQ_{lj}.

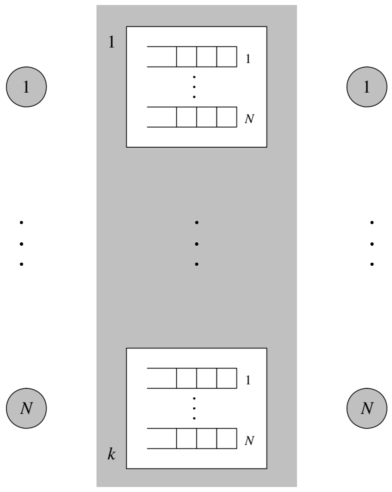

Fig. 3. Load balanced switch

Since every output queued switch runs at a speedup of N/k, the internal lines connecting the input and output ports to the k switches need only run at $1/k$ the speed of the external input and output lines. Alternatively, one could describe the operation of a load balanced switch in *an enlarged time step* as follows:

Load Balancing

- input i receives up to k packets on the input line
- [load balancing] input i sends at most one packet to every switch until all packets are sent
- output j retrieves at most one packet from VOQ_{lj} for all $l = 1 \ldots k$
- output j delivers up to k packets on the output line

The compromise between the two previous architectures is manifested by a reduction in speedup from N to N/k with possibly eliminating it totally if $N \leq k$, and by a replacement of the global matching algorithm with a local load balancing algorithm at every port (almost no scheduling at all).

However, load balancing algorithms may reorder packets belonging to the same flow. For instance, suppose that two packets destined to output j arrive at input i in the same time step (up to k packets may now arrive at input i in every time step). The two packets will be placed in different switches and may experience different delays through their respective VOQs of possibly different lengths and, therefore, may reach output j unordered by an arbitrary amount. Although at the conceptual level reordering is not necessarily a problem, some network applications like TCP do no perform well when packets of the same flow are reordered. Therefore, we generally insist not to reorder packets belonging to the same flow.

The problem of reordering is traditionally solved by adding input and/or output buffers to the architecture of Figure 3. For instance, one approach is to route each flow through at most one switch [16]. Such an approach requires input buffers (an input cannot send two packets to the same switch in one time step). Although it avoids reordering, this approach results in a loss of throughput due to small capacities of different switches that remain unused; and nevertheless, non of them is large enough by itself to route a single flow [16].

Since, as explained above, reordering may occur when two VOQs for the same output (in different switches) have different lengths, another approach to avoid reordering is to ensure that for each output j, all VOQ_{lj} receive the same number of packets for $l = 1 \ldots k$. This approach is used in [17] for a switch consisting of k parallel input queued switches. It is also described in [10] for the load balanced switch as the UFS (Uniform Frame Spreading) algorithm and proved to achieve 100% throughput. With UFS, whenever input i sends packets, it sends exactly k packets of the same flow, one to every switch. Therefore, all VOQs for a given output receive the same number of packets in every time step. This approach also requires buffering at the input ports: packets destined to a given output are buffered at the input until there are k of them. However, this approach might lead to starvation. For instance, if input i receives a one and only packet for output j, then this packet never leaves the input buffer, and hence is never received by output j. Therefore, input i is starved.

Yet another approach is to allow a limited amount of reordering in the switch (with the help of input buffers) and later correct for misordered packets at the output by adding output buffers. The work of [7] is the first of this kind and presents the Parallel Packet Switching algorithm PPS, which mainly sends packets of the same flow from the input buffer to the k switches in a round robin order. The algorithm of [2] is almost identical to the PPS algorithm; however, [2] also presents a variation on PPS using an Earliest Deadline First queuing algorithm EDF on the VOQs, which requires the VOQs to be non FIFO. The work of [9] presents an enhancement over that of [2] at the expense of using even more complicated queuing mechanism in the switches (again requiring non FIFO VOQs). The work of [10] presents the Full Ordered Frames First algorithm, FOFF. The FOFF algorithm is identical to UFS with the following exception: FOFF allows the input to send less than k packets while UFS does not, but of course only when no flow has at least k packets in the input buffer [1].

As this paper will show, adding input and/or output buffers to the architecture of Figure 3 is a natural and inevitable outcome of the research on load balancing. The paper is organized as follows: For completeness, Section II illustrates some basic load balancing algorithms (with reordering). Section III provides some preliminary definitions and prepares for the following sections. The main contribution of the paper lies in Sections IV and V. Section IV proves an impossibility result: there is no

[1]Therefore, FOFF does not starve an input, but can still starve a flow.

load balancing algorithm that achieves 100% throughput and no reordering for a load balanced switch without buffers. Section V proves a possibility result: there is a load balancing algorithm that achieves 100% throughput and limited reordering for a load balanced switch without buffers. This latter result implies that there is a load balancing algorithm that achieves 100% throughput and no reordering for a load balanced switch with output buffers only; hence, establishing a theoretical evidence that input buffers are not necessary. We conclude in Section VI.

II. Basic Load Balancing (With Reordering)

One possible load balancing algorithm is based on a round robin approach:

Round Robin Load Balancing (t is the time step)

- input i receives up to k packets on the input line
- input i sends its packets sorted by their outputs in a round robin order starting at switch $[(t-1) \bmod k]+1$
- output j retrieves at most one packet from VOQ_{lj} for all $l = 1 \dots k$
- output j delivers up to k packets on the output line

Therefore, input i sends all packets for output 1, then all packets for output 2, etc... starting at switch $[(t-1) \bmod k]+1$ and continuing in a round robin order of the switches. For now assume that an admissible traffic satisfies $\sum_{i=1}^{N} r_{ij} < k$ where r is the rate matrix. Using a proof technique similar to [1], we can prove that the round robin load algorithm guarantees 100% throughput for any admissible traffic that is stationary and weakly mixing.

While the weakly mixing requirement is stronger than ergodicity needed for an output queued switch, this requirement can be relaxed by using randomization.

Randomized Load Balancing

- input i receives up to k packets on the input line
- input i sends its packets to the k switches randomly (at most one packet per switch)
- output j retrieves at most one packet from VOQ_{lj} for all $l = 1 \dots k$
- output j delivers up to k packets on the output line

Again, using a proof technique similar to [1] we can prove that the randomized load balancing algorithm achieves 100% throughput for any admissible traffic that is stationary and ergodic.

III. Preliminaries

Let $a(t)$ be an $N \times N$ matrix where $a_{ij}(t)$ is the number of packets received at input i for output j at time t. Let $A(t) = \sum_{s=1}^{t} a(s)$ be the matrix representing the cumulative number of packets up to time t.

Definition 1—Admissible traffic: A traffic is admissible iff:

$$\lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{N} A_{ij}(t) \le k$$

Definition 2—Non-bursty traffic: A traffic is non-bursty iff there exists an r_j for all $j = 1 \dots N$ such that:

$$|\sum_{i=1}^{N} A_{ij}(t) - r_j t| \le B, \forall\, t$$

where B is a burst constant (with respect to t) [2].

Definition 3—Reordering: A switch reorders packets iff two packets of the same flow are received by their output port in the wrong order in different time steps.

We adopt the following operational notion of 100% throughput:

Definition 4—100% throughput: A switch achieves 100% throughput iff the number of packets served by the switch is within a constant from the number of packets served by an output queued switch, infinitely many times.

Lemma 1: Let $D(t)$ be an $N \times N$ matrix representing the cumulative number of packets of an admissible non-bursty traffic served by an output queued switch. Then $||A(t) - D(t)|| \le 2NB$, where B is the burst constant.

According to Lemma 1, whenever the traffic is admissible and non-bursty, we can use the fact that the number of packets remaining in the output queued switch at any time is upper bounded by a constant. Therefore, for a load balanced switch to achieve 100% throughput under such traffic, the number of packets remaining in the switch will have to be upper bounded by a constant infinitely many times (see Definition 4). We end this section with a proof for Lemma 1.

Proof: Consider an output queued switch where one unit of service can serve up to k packets. Let $A_j(t)$ be the cumulative number of packets for output j. Similarly, let $D_j(t)$ be the number of packets in $A_j(t)$ that are served by the output queued switch. Therefore, $A_j(t) = \sum_i A_{ij}(t)$ and $D_j(t) = \sum_i D_{ij}(t)$. Since the output queued switch is work conserving (i.e, non-idling), then ([3] page 7)

$$D_j(t) = \min_{0\le s\le t} [A_j(s) + k(t-s)]$$

Let $s \le t$. By the burst condition,

$$\begin{aligned} A_j(t) &\le r_j t + B \\ &= (r_j - k)t + B + kt \end{aligned}$$

By the admissibility condition, $r_j - k \le 0$; otherwise the burst condition implies that $\lim_{t\to} \frac{1}{t} A_j(t) =$

[2]Both Definitions 1 and 2 assume that the limit of $\frac{1}{t}\sum_i A_{ij}(t)$ exists. Alternatively, we can relax both definitions and combine them into one: there exists an $r_j \le k$ such that $\sum_i A_{ij}(t_2) - \sum_i A_{ij}(t_1) - r_j(t_2 - t_1) \le B$ for any $t_1 \le t_2$. This change does not affect Lemma 1. In either case, if the rate matrix $r = \lim_{t\to\infty} \frac{1}{t} A(t)$ exists, $\sum_{i=1}^{N} r_{ij} \le k$ and $\sum_{j=1}^{N} r_{ij} \le k$ (because $\sum_{j=1}^{N} a_{ij} \le k$).

$\lim_{t\to} \frac{1}{t}\sum_{i=1}^{N} A_{ij}(t) > k$ and the traffic is not admissible. Therefore, since $s \le t$,

$$\begin{aligned} A_j(t) &\le (r_j - k)s + B + kt \\ &= (r_j s - B) + k(t-s) + 2B \end{aligned}$$

By the burst condition again, $r_j s - B \le A_j(s)$; therefore,

$$A_j(t) \le [A_j(s) + k(t-s)] + 2B$$

Since $s \le t$ is arbitrary,

$$\begin{aligned} A_j(t) &\le \min_{0\le s\le t}[A(s) + k(t-s)] + 2B \\ &= D_j(t) + 2B \end{aligned}$$

Therefore,

$$A_j(t) - D_j(t) \le 2B$$

Summing over $j = 1 \ldots N$ yields $||A(t) - D(t)|| \le 2NB$. ■

IV. The Impossibility Result

In this section we prove that it is not possible to achieve 100% throughput and no reordering without buffers. We prove this impossibility result by forcing any load balancing algorithm that does not reorder to make some output j serve less than k packets infinitely many times, while generating an admissible and non-bursty traffic that satisfies $\sum_i A_{ij}(t) = kt$ infinitely many times. This makes the number of packets remaining in the switch $||A(t) - D(t)||$ an increasing function of t, and hence by Lemma 1 and Definition 4, the algorithm will not achieve 100% throughput.

We denote by $|VOQ_{lj}|_t$ the length of VOQ_{lj} at the beginning of time step t. Let $max_j(t) = \max_l |VOQ_{lj}|_t$ and $min_j(t) = \min_l |VOQ_{lj}|_t$ and $\Delta_j(t) = max_j(t) - min_j(t)$. Therefore, we denote by a largest (smallest) VOQ_{lj} at time t the one for which l is a maximizing (minimizing) argument for $max_j(t)$ ($min_j(t)$).

Definition 5—The unbalancer: Input i *unbalances* output j at time t if i sends a packet to a largest VOQ_{lj} at time t among a total of less than k packets to all VOQs of output j.

Lemma 2: If output j serves k packets from its VOQs in every time step, and if the VOQs of output j receive packets only from input i, and if i unbalances j more than $(k-2)\Delta_j(t)$ times in $[t, t']$, but otherwise sends k packets to the VOQs of output j, then $\Delta_j(t') - \Delta_j(t) \ge 1$.

Proof: Define $c(t) = \sum_l (max_j(t) - |VOQ_{lj}|_t)$, then $c(t) \ge max_j(t) - min_j(t) = \Delta_j(t)$. If input i sends k packets to the VOQs of output j at time t, then $c(t+1) = c(t)$ (every VOQ receives one packet and delivers one packet). If input i unbalances output j at time t, then $c(t+1) \ge c(t) + 1$ (a largest VOQ_{lj} receives one packet while at least one other VOQ does not, and every VOQ delivers one packet). Therefore $c(t)$ increases by at least 1 every time i unbalances j.

$$\begin{aligned} c(t') &\ge c(t) + (k-2)\Delta_j(t) + 1 \\ c(t') &\ge \Delta_j(t) + (k-2)\Delta_j(t) + 1 \\ &= (k-1)\Delta_j(t) + 1 \end{aligned}$$

This implies that at time t', there exists at least one VOQ_{lj} such that $max_j(t') - |VOQ_{lj}|_{t'} \ge \Delta_j(t) + 1$ (pigeonhole principle). Therefore, $\Delta_j(t') - \Delta_j(t) \ge 1$. ■

To see the implication of Lemma 2, consider $\Delta_j(t_0) \ge 0$. By using an unbalancer input i, we can make $\Delta_j(t_1) \ge 1$, then $\Delta_j(t_2) \ge 2$, etc... where $t_0 < t_1 < t_2 < \ldots$. Therefore, we can create an arbitrary difference in the length of VOQs of output j.

Lemma 2 provides the main mechanism for constructing our adversarial traffic. Recall that we are interested in showing that some output j will serve less than k packets infinitely many times (provided that the load balancing algorithm does not reorder). We will show a scenario that forces this once, but then the whole scenario can be repeated, yielding the desired behavior.

A. The basic idea

Here's the scenario that we plan to construct. Assume without loss of generality that every output serves k packets from its VOQs in every time step (if not, then we are done). Let every input receive a total of k packets in every time step. Therefore, every input sends k packets in every time step. Based on Lemma 2, and by using unbalancers (see Definition 5), let some output j have a large enough $\Delta_j(t)$, say $\Delta_j(t) \ge 2$. This scenario is illustrated in Figure 4 below.

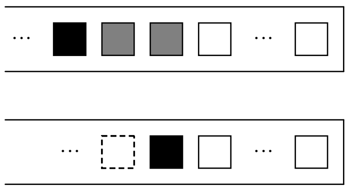

Fig. 4. A $\Delta_j(t) = 2$ for VOQ_{lj}

The gray packets in Figure 4 illustrate the difference in length among two VOQs of output j. By the end of time step t let there be only one input, say i, sending packets to output j. Input i will send k packets to the VOQs of output j in every time step. Therefore, the two VOQs above will receive one packet each at time $t+1$. These are shown in black in Figure 4. But the one empty position shown in the lower VOQ of Figure 4 will eventually contain a packet that arrived to input i after the black packets. If output j continues to serve k packets from its VOQs in every time step, it will eventually misorder the black packet in the upper VOQ of Figure 4 (a later packet would be served in a previous time step). Therefore, output j will have to serve less than k packets at some time.

Of course the question now is how to construct the above scenario in a way that it can be repeated infinitely many times, and such that the traffic is admissible and non-bursty and satisfies $\sum_i A_{ij}(t) = kt$ infinitely many times.

We start by assuming that N is large enough compared to k. N will be a function of k; however, to keep the illustration simple, we will not attempt to calculate an exact value for N.

B. The main mechanism

The N input and output ports will form pairs $(i_1, j_1), \ldots, (i_N, j_N)$. Input i_x receives k packets for output j_x in every time step. Occasionally, input i_x may receive a packet for output j_y (and hence only $k-1$ packets for output j_x), where $x \neq y$; however, only one such packet may be received in a given time step; we call such packet an *orphan* packet. We will explicitly state when an orphan packet is received; otherwise, the default is that input i_x receives k packets for output j_x in every time step.

We will show how to force some output j to have $\Delta_j(t) \geq 2$, then the scenario of Figure 4 follows immediately from the main mechanism described above. Since N is large enough, there is a large number of outputs, say $M \leq N$, that have their largest VOQ in the same switch (pigeonhole principle). Without loss of generality, assume this is switch 1 and that these M outputs correspond to the pairs $(i_1, j_1), \ldots, (i_M, j_M)$. We can also assume that M is itself large enough such that $M \geq 2k$. We will engage these M pairs in a tournament of games at the end of which there will be a winner pair (i, j) with $\Delta_j(t) \geq 2$. We first explain a single game.

C. The game

The game consists of $P = (k-2)h + 2$ player pairs at time t, indexed from 1 to P without loss of generality $(i_1, j_1), \ldots, (i_P, j_P)$, such that $\Delta_{j_x}(t) \geq h$ for all $x = 1 \ldots P$, for some h.

Recall that by default input i_x receives (and hence also sends) k packets for output j_x in every time step. The objective of the game is to end up with a pair (i, j) such that $\Delta_j(t') \geq h+1$ for some $t' \geq t$. If any of the pairs satisfies this at time t, then we are done. Otherwise, let (i_1, j_1) be the coordinator of the game.

Input i_1 receives $k-1$ packets for j_1 and an orphan packet p_2 for j_2 at time $t+1$. If p_2 is sent to switch 1 (where all P outputs have their largest VOQ), then pair (i_2, j_2) will have $\Delta_{j_2}(t+1) \geq h+1$ and we are done (the packet is sent to the largest VOQ). If p_2 is sent to some other switch, then i_1 unbalances j_1 (i_1 sends less than k packets to the VOQs of output j_1, with one packet being sent to switch 1). If i_1 is an unbalancer at time $t+1$, then i_1 receives $k-1$ packets for j_1 and an orphan packet p_3 for j_3 at time $t+2$. If p_3 is sent to switch 1, then pair (i_3, j_3) will have $\Delta_{j_3}(t+2) \geq h+1$ and we are done. If p_3 is sent to some other switch, then i_1 unbalances j_1 for a second time. If i_1 is an unbalancer at time $t+2$, then i_1 receives $k-1$ packets for j_1 and an orphan packet p_4 for j_4 at time $t+3$, and the same process repeats.

Eventually at time $t' \geq t$, either we have some output $j \neq j_1$ such that $\Delta_j(t') \geq h+1$, or i_1 unbalances j_1 $P-1 = (k-2)h+1$ times. In the latter case, $\Delta_{j_1}(t') \geq h+1$ by Lemma 2. Therefore, we have a winner pair (i, j) at some time $t' \geq t$ with $\Delta_j(t') \geq h+1$.

Note that when the game ends with pair (i, j) as the winner, output j maintains its largest VOQ in switch 1 for future times. So the winner can participate in another game as described below.

D. The tournament

If the tournament ends at time t, we would like to declare a winner pair (i, j) with $\Delta_j(t) \geq 2$. Therefore, at the last stage of the tournament, we need a game with $P = (k-2)(1) + 2 = k$ players.

But for the game to work, all k players must start with $\Delta_j(t') \geq 1$ for some $t' \leq t$. To guarantee the existence of such players at time t', we make each player the winner of a previous game. Looking at one player now, we want $\Delta_j(t') \geq 1$; hence, we need a previous game with $(k-2)(0) + 2 = 2$ players. Therefore, $M = 2k$ players are enough to construct the tournament and obtain a winner pair (i, j) with $\Delta_j(t) \geq 2$. Note that the existence of M players is guaranteed by the choice of N to be large enough to find M pairs with all outputs having their largest VOQ in the same switch.

The construction of the tournament makes it possible to reach the scenario of Figure 4, which in turn implies that some output j will serve less than k packets at some time t. Therefore, it would be tempting to wait until that happens, and then repeat the tournament infinitely many times. However, there is an important issue that still needs to be considered: some flows may receive more (or less) than k packets in some time steps (because of orphan packets); therefore, we risk not having an admissible and non-bursty traffic by simply repeating the tournament infinitely many times.

Fortunately, this issue can be dealt with easily. Note that the number of orphan packets generated during a tournament is bounded[3]. Therefore $|\sum_i A_{ij}(t) - kt| \leq B$ for some constant B (with respect to t).

Before repeating the tournament, the adversary performs a cleanup phase in which an output j_x that received an excess of packets will receive less than k packets from i_x for a number of time steps until it makes $\sum_i A_{ij_x}(t) = kt$ for some time t. Similarly, an output j_x that received a deficit of packets will receive more than k packets for a number of time steps until it makes $A_{ij_x}(t) = kt$ for some time t. This is possible by using more inputs and orphan packets (but only a bounded number of both).

E. The adversarial traffic

Assuming the main mechanism described in Section IV.B as our default traffic pattern, the adversarial traffic will be as follows:

```
Adversary
**while** (TRUE)
    - identify $M = 2k$ pairs $(i, j)$ with all outputs
      having their largest $VOQ$ in the same switch
    - construct the tournament
    - wait until some output $j$ serves less than $k$ packets
    - cleanup (make traffic satisfy $\sum_i A_{ij}(t) = kt \; \forall \; j = 1 \ldots N$
```

[3]This is because we have a bounded number of games in a tournament, and the number of orphan packets generated in a single game is at most $P-1$ where P is the number of players in that game. An upper bound on the number of orphan packets in a tournament can be computed as $M - 1 = 2k - 1$.

The adversarial traffic described above forces some output to serve less than k packets infinitely many times. Since the traffic satisfies $|\sum_i A_{ij}(t) - kt| \leq B \; \forall \; t$ and $\sum_i A_{ij}(t) = kt$ infinitely many times, for all $j = 1 \ldots N$, we have that $||A(t) - D(t)||$ is an increasing function of time, hence the following result:

Theorem 1—Impossibility: It is impossible to achieve 100% throughput and no reordering for a load balanced switch without buffers (when $N >> k \geq 2$).

Proof: By using the adversarial traffic described above and Lemma 1 and Definition 4 of 100% throughput. ∎

V. The Possibility Result

In this Section we present a load balancing algorithm that achieves 100% throughput and limited reordering [4] without buffers. This implies that it is possible to achieve 100% throughput and no reordering with the use of output buffers only. To our knowledge, this fact has not been noted in any prior work on load balancing in a switch.

Let $z_{lj}(t)$ be the number of packets sent to VOQ_{lj} at time t and $Z_{lj}(t) = \sum_{s=1}^{t} z_{lj}(t)$ be the cumulative number of packets sent to VOQ_{lj}. The basic approach for designing our load balancing algorithm relies on the following two observations for the architecture of Figure 3 (they can be obtained by generalizing Appendices D and E of [10]).

- Observation I (no output buffer): if the load balancing algorithm guarantees that $|Z_{lj}(t) - Z_{l'j}(t)|$ is bounded at any time t for all $1 \leq l, l' \leq k$ and all $j = 1 \ldots N$, and if all VOQs are work conserving, then the algorithm achieves 100% throughput and limited reordering [5].
- Observation II (adding output buffers): for the same setting above, it is possible to add bounded size output buffers, and make the load balancing algorithm achieve 100% throughput with no reordering [6].

Therefore, all we need is to show the existence of a load balancing algorithm that guarantees $|\frac{1}{k}\sum_i A_{ij}(t) - Z_{lj}(t)|$ is bounded at all times for all $l = 1 \ldots k$, and all $j = 1 \ldots N$.

The load balancing algorithm will conceptually consist of k algorithms Send-A-Packet$_1$, ..., Send-A-Packet$_k$ that are performed sequentially in turn in one time step at every input port. Send-A-Packet$_1$ chooses a packet among all the packets received at input i and sends it to the first switch. Send-A-Packet$_2$ then chooses a packet among the remaining packets at input i and sends it to the second switch; and so on until all packets are sent. Therefore, the algorithms Send-A-Packet$_l$ for $l = 1 \ldots k$ experience different traffic patterns. For instance, Send-A-Packet$_1$ experiences the traffic described by $a(t)$ and $A(t)$. In general, Send-A-Packet$_l$ experiences the traffic described by $a^l(t)$ and $A^l(t)$, where:

$$\sum_{i=1}^{N} a_{ij}^{l}(t) = \sum_{i=1}^{N} a_{ij}(t) - \sum_{x=1}^{l-1} z_{xj}(t)$$

$$\sum_{i=1}^{N} A_{ij}^{l}(t) = \sum_{s=1}^{t} a_{ij}^{l}(s) = \sum_{i=1}^{N} A_{ij}(t) - \sum_{x=1}^{l-1} Z_{xj}(t)$$

where an empty summation is assumed to be zero.

We will first show that under this conceptual view of the load balancing algorithm, $|\frac{1}{k-l+1}\sum_i A_{ij}^l(t) - Z_{lj}(t)| < D$ at all times for all $l = 1 \ldots k$ and all $j = 1 \ldots N$, for some constant D (with respect to time). Then we show that $|\frac{1}{k}\sum_i A_{ij}(t) - Z_{lj}(t)|$ is bounded at all times for all $l = 1 \ldots k$, and all $j = 1 \ldots N$. Without loss of generality, we make the following assumptions:

- **there is only one input port** i: for whatever bound D we compute under this assumption, we can simply multiply D by N to account for all input ports.
- **input** i **receives** k **packets in every time step**; therefore, Send-A-Packet$_l$ for every $l = 1 \ldots k$ always finds a packet to send: if not, we can always fill the remaining empty packets with packets for a fictitious output [7]. Input i will receive therefore packets for $N + 1$ outputs.
- let $O_l(t)$ be a subset of outputs defined as follows: $a_{ij}^l(t) \neq 0$ iff $j \in O_l(t)$. Obviously $O_l(t) \in 2^{\{1,\ldots,N\}}$ which is the power set of $\{1, \ldots, N\}$, and $|O_l(t)| \leq k - l + 1$. **For Send-A-Packet$_l$, $l = 1 \ldots k$, we consider only the times t when $O_l(t) = O_l$ for a particular subset of outputs O_l** (i.e. $a(t)$, and consequently other quantities as well, are defined for those times only) [8]: for whatever bound D we compute under this assumption, we can simply multiply D by 2^{N-1} to account for all possible subsets of outputs, since every output belongs to exactly 2^{N-1} elements of $2^{\{1,\ldots,N\}}$.

For all $j \in O_l$, let:

$$c_{lj}(t) = \frac{1}{k-l+1} A_{ij}^l(t) - Z_{lj}(t)$$

$$c_l^*(t) = \max_{j \in O_l} c_{lj}(t)$$

$$c_{l*}(t) = \min_{j \in O_l} c_{lj}(t)$$

$$g_{lj}(t) = \frac{1}{k-l+1} a_{ij}^l(t)$$

Then

$$c_{lj}(t) = c_{lj}(t-1) + g_{lj}(t) - z_{lj}(t)$$

[4]If two packets p_1 and p_2 belong to flow (i, j), and p_1 arrives to input i before p_2, then output j will retrieve p_1 at most a bounded number of time steps after p_2.

[5]Starvation cannot occur because there are no buffers and all VOQs are FIFO.

[6]By observation I, for every packet in the output buffer, there will be a time when delivering that packet cannot violate the packet order; therefore, the algorithm can easily avoid starvation.

[7]Packets to the fictitious output are scheduled by the load balancing algorithm but not actually sent to any physical VOQ.

[8]Therefore, every Send-A-Packet$_l$ can be viewed as 2^N separate algorithms running in parallel, each responding to one particular subset of outputs. However, we still use t and $t + 1$ to denote two consecutive time steps for a given O_l.

By the first assumption (there is only one input port i), $z_{lj}(t)$ is either 0 or 1. Moreover, by all three assumptions, we have $\sum_{j\in O_l} a^l_{ij}(t) = k-l+1$, $\sum_{j\in O_l} A^l_{ij}(t) = (k-l+1)t$, and $\sum_{j\in O_l} Z_{lj}(t) = t$. Therefore:

$$\sum_{j\in O_l} c_{lj}(t) = 0$$

$$\sum_{j\in O_l} g_{lj}(t) = 1$$

$$\frac{1}{k-l+1} \le g_{lj}(t) \le 1 \,\forall\, j \in O_l$$

We state the following lemma:

Lemma 3: If $c^*_l(t) \le 1 + \sum_{j\in O_l} c_{lj}(t) g_{lj}(s)$, then $\sum_{j\in O_l} c^2_{lj}(t) < k^3$.

Proof:

$$\begin{aligned} c^*_l(t) &\le 1 + \textstyle\sum_{j\in O_l} c_{lj}(t) g_{lj}(s) \\ &\le 1 + c^*_l(t)\tfrac{k-l}{k-l+1} + c_{l*}(t)\tfrac{1}{k-l+1} \end{aligned}$$

The second inequality follows because $\sum_{j\in O_l} g_{lj}(s) = 1$ and $\frac{1}{k-l+1} \le g_{lj}(s) \le 1 \,\forall\, j \in O_l$. Therefore,

$$c^*_l(t) - c_{l*}(t) \quad \le \quad k-l+1$$

Since $\sum_{j\in O_l} c_{lj}(t) = 0$, $|c_{lj}(t)| < k-l+1$ for all $j = 1\ldots N$. Therefore, $\sum_{j\in O_l} c^2_{lj}(t) < |O_l|(k-l+1)^2 \le (k-l+1)^3 \le k^3$. ■

We now describe our Send-A-Packet algorithms (see footnote 8):

```
Send-A-Packet$_l(t)$
  for every $j \in O_l$
    do if $c_{lj}(t-1) = c^*_l(t-1)$
      then $j \mapsto_t l$
        return
```

where $j \to_t l$ denotes that input i sends a packet for output j to switch l at time t (i.e. to VOQ_{lj}). Basically, in every time step of Send-A-Packet$_l$, input i determines an output $j \in O_l$ for which $c_{lj}(t-1)$ is maximum and sends a packet for that output to switch l.

Lemma 4: The Send-A-Packet$_l$ algorithm guarantees that $|c_{lj}| < \sqrt{k^3+2}$ at all times t for all $j = 1\ldots N$.

Proof:

$$\sum_{j\in O_l} c^2_{lj}(t) - \sum_{j\in O_l} c^2_{lj}(t-1) =$$

$$\sum_{j\in O_l} [c_{lj}(t-1) + g_{lj}(t) - z_{lj}(t)]^2 - \sum_{j\in O_l} c^2_{lj}(t-1) =$$

$$\sum_{j\in O_l} [g_{lj}(t) - z_{lj}(t)]^2 + 2\sum_{j\in O_l} c_{lj}(t-1)[g_{lj}(t) - z_{lj}(t)]$$

Since by the Send-A-Packet$_l$ algorithm, $z_{lj}(t) = 1$ iff $c_{lj}(t-1) = c^*_l(t-1)$,

$$\sum_{j\in O_l} [g_{lj}(t) - z_{lj}(t)]^2 < 2$$

$$\sum_{j\in O_l} c_{lj}(t-1)[g_{lj}(t) - z_{lj}(t)] = \sum_{j\in O_l} c_{lj}(t-1) g_{lj}(t) - c^*_l(t-1)$$

Therefore,

$$\sum_{j\in O_l} c^2_{lj}(t) - \sum_{j\in O_l} c^2_{lj}(t-1) < 2(1-\gamma(t)) < 2$$

where $\gamma(t) = c^*_l(t-1) - \sum_{j\in O_l} c_{lj}(t-1) g_{lj}(t) \ge 0$. Now consider the first time t such that $\sum_{j\in O_l} c^2_{lj}(t) \ge k^3+2$; therefore, $\sum_{j\in O_l} c^2_{lj}(t) > \sum_{j\in O_l} c^2_{lj}(t-1)$. By the above inequality, $\sum_{j\in O_l} c^2_{lj}(t-1) > k^3$. By Lemma 3, $\gamma(t) > 1$. By the above inequality, $\sum_{j\in O_l} c^2_{lj}(t) < \sum_{j\in O_l} c^2_{lj}(t-1)$, a contradiction. Therefore, $\sum_{j\in O_l} c^2_{lj}(t) < k^3+2$ at all times t. Consequently, $|c_{lj}(t)| < \sqrt{k^3+2}$ at all times t for all $j = 1\ldots N$. ■

Therefore, under the assumptions we mentioned earlier, $D = \sqrt{k^3+2}$. In order to relax all assumptions, we set $D = N2^{(N+1)-1}\sqrt{k^3+2} = N2^N\sqrt{k^3+2}$, where the multiplication by N is to account for all inputs, the replacement of N by $N+1$ is to account for a fictitious output, and the multiplication by $2^{(N+1)-1} = 2^N$ is to account for all subsets of $\{1,\ldots,N+1\}$ for $O_l(t)$.

Therefore, $|\frac{1}{k-l+1}\sum_i A^l_{ij}(t) - Z_{lj}(t)| < N2^N\sqrt{k^3+2}$ at all times for all $l = 1\ldots k$ and all $j = 1\ldots N$. Consider the following load balancing algorithm:

```
Load-Balance$(t)$
  for $l \leftarrow 1$ to $k$
    do Send-A-Packet$_l(t)$
```

The Load-Balance algorithm runs in $O(k^2)$ time at every input port. This is not necessarily an efficient algorithm for load balancing; however, it provides evidence for the following theorem.

Theorem 2—Possibility: There exists a load balancing algorithm that achieves 100% throughput and limited reordering for a load balanced switch without buffers, and no reordering with the addition of output buffers.

Proof: Based on Observations I and II, it is enough to prove that the Load-Balance algorithm guarantees $|\sum_i \frac{1}{k} A_{ij}(t) - Z_{lj}(t)|$ is bounded at all times for all $l = 1\ldots k$ and all $j = 1\ldots N$. We prove it by induction on l.

Base case: for $l = 1$ we have $|\frac{1}{k}\sum_i A^1_{ij}(t) - Z_{1j}(t)| < N2^N\sqrt{k^3+1}$ for all $j = 1\ldots N$. Therefore, $|\frac{1}{k}\sum_i A_{ij}(t) - Z_{1j}(t)| < N2^N\sqrt{k^3+1}$ for all $j = 1\ldots N$.

Induction step: given that $|\frac{1}{k}\sum_i A_{ij}(t) - Z_{xj}(t)|$ is bounded for all $x = 1\ldots l-1$ and all $j = 1\ldots N$, we prove that

$|\frac{1}{k}\sum_i A_{ij}(t) - Z_{lj}(t)|$ is bounded for all $j = 1 \ldots N$. Since $|\frac{1}{k}\sum_i A_{ij}(t) - Z_{xj}(t)|$ is bounded for all $x = 1 \ldots l-1$, then

$$|\frac{l-1}{k}\sum_i A_{ij}(t) - \sum_{x=1}^{l-1} Z_{xj}(t)|$$

is bounded. Moreover, we have that

$$|\frac{1}{k-l+1}\sum_i A^l_{ij}(t) - Z_{lj}(t)|$$

is bounded. Since $\sum_i A^l_{ij}(t) = \sum_i A_{ij}(t) - \sum_{x=1}^{l-1} Z_{lj}(t)$, we have that

$$|\frac{1}{k-l+1}(\sum_i A_{ij}(t) - \sum_{x=1}^{l-1} Z_{xj}(t)) - Z_{lj}(t)|$$

is bounded. The above expression is equal to the following:

$$|\frac{1}{k}\sum_i A_{ij}(t) - Z_{lj}(t) + \frac{1}{k-l+1}(\frac{l-1}{k}\sum_i A_{ij}(t) - \sum_{x=1}^{l-1} Z_{xj}(t))|$$

Therefore, $|\frac{1}{k}\sum_i A_{ij}(t) - Z_{lj}(t)|$ is bounded for all $j = 1 \ldots N$. Since the inductive step is needed only $k-1$ times, $|\frac{1}{k}\sum_i A_{ij}(t) - Z_{lj}(t)|$ is bounded at all times for all $l = 1 \ldots k$ and all $j = 1 \ldots N$.

■

VI. Conclusion

We proved the theoretical impossibility of achieving 100% throughput and no reordering for a load balanced switch without buffers. We also proved that it is possible to achieve 100% throughput and limited reordering for a load balanced switch without buffers, by providing a new load balancing algorithm. Therefore, this algorithm achieves 100% throughput and no reordering with the use of output buffers only. Although this algorithm is not necessarily the most efficient way of load balancing, it provides a theoretical evidence that input buffers are not necessary.

References

[1] C.-S. Chang, D.-S. Lee, T.-S. Jou, *Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering.* Computer Communications, Vol. 25, No. 6, pp. 611-622, 2002.

[2] C.-S. Chang, D.-S. Lee, C.-M. Lien, *Load balanced Birkhoff-von Neumann switches, part II: multi-stage buffering.* Computer Communications, Vol. 25, No. 6, pp. 623-634, 2002.

[3] C.-S. Chang, *Performance Guarantees in Communication Networks.*, Springer-Verlag, New York, 2000.

[4] A. Charny, P. Krishna, N. Patel, R. Simcoe, *Algorithms for providing bandwidth and delay guarantees in input buffered crossbars with speedup.* Proceedings of 6^{th} International Workshop on Quality of Service, IWQOS 98, pp. 235-44, May 1998.

[5] S.-T. Chuang, A. Goel, N. McKeown, B. Prabhakar, *Matching output queuing with combined input output queued switches.* IEEE Journal of Selected Areas in Communication 17(6), pp. 1030-39, June 1999.

[6] J. G. Dai, B. Prabhakar, *The throughput of data switches with and without speedup.* IEEE/ACM INFOCOM 2000, pp. 556-64.

[7] S. Iyer, N. McKeown, *Making parallel packet switches practical.* Proceedings of IEEE INFOCOM 2001, pp. 1680-87, March 2001.

[8] A. Kam, K.-Y. Siu, R. Barry, *A cell switching WDM broadcast LAN with bandwidth guarantee and fair access.* IEEE/OSA Journal of Lightwave Technology 16(2), pp. 2265-80, December 1998.

[9] I. Keslassy, N. McKeown Maintaining packet order in two-stage switches. IEEE Infocom, 2002, New York, NY, June 2002.

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

[11] P. Krishna, N. S. Patel, and A. Charny, *On the speedup requirement for work conserving crossbar switches.* IEEE Journal of Selected Areas in Communication 17(6), pp. 1057-69, June 1999.

[12] N. McKeown, V. Anantharam, J. Walrand, *Achieving 100% throughput in an input queued switch.* IEEE INFOCOM 96, vol. 1, pp. 296-302, March 1996.

[13] N. Mckeown, *The iSLIP scheduling algorithm for input queued switches.* IEEE/ACM Transactions on Networking, 7(2), pp. 188-201, April 1999.

[14] A. Mekkittikul and N. McKeown, *A starvation-free algorithm for achieving 100% throughput in an input queued switch.* Proceeding of the International Conference on Computer Communication and Networking, ICCCN 96, pp. 226-231, October 1996.

[15] A. Mekkittikul, N. McKeown, *A practical scheduling algorithm to achieve 100% throughput in input queued switches.* IEEE INFOCOM 1998, vol. 2, pp. 792-99, March-April 1998.

[16] S. Mneimneh, K.-Y. Siu, Scheduling unsplittable flows using parallel switches. ICC2002, New York, USA, April 2002.

[17] S. Mneimneh, V. Sharma, K.-Y. Siu, Switching using parallel input-output queued switches. IEEE/ACM Transactions on Networking 10(5), 2002.

[18] S. Mneimneh, K.-Y. Siu, On achieving throughput in an input queued switch. IEEE/ACM Transactions on Networking 11(5), 2003.

[19] A. Baldini, D. Bergamasco, D. Blumenthal, D. Chiou, S.-T. Chuang, V. De Feo, P. Donner, S. Fraser, Y. Ganjali, P. Giaccone, I. Keslassy, M. Kodialam, F. Mattus, N. McKeown, D.-S. Lee, S. Mneimneh, M. Neely, A. Smiljanic, B. Towles, R. Zhang-Shen, M. Zirngibl, *Stanford Workshop on Load Balancing.* Stanford Universiry, CA, May 19 2004. http://yuba.stanford.edu/lb/ .