Speedup required for achieving 100% throughput with FPWWFA

Shang-Tse Chuang and Siva Gaggara

1 Abstract

High-speed switch architectures are more frequently designed with single-stage non-blocking fabrics. This trend has been fueled by the need for faster line rates which in turn have placed a greater burden on the memory bandwidth requirements. It has been shown that speedup can help provide throughput guarantees. In this paper, we investigate throughput of a switch using FPWWFA, a practical maximal matching algorithm. We have observed that a speedup of 4/3 is the minimum speedup required to achieve 100% throughput for any non-uniform Bernoulli traffic. This result will allow switches to put less stringent requirements on the memory.

2 Introduction

Many switch architects require a switch fabric to be capable of delivering 100% throughput for input traffic - uniform or non-uniform, and that does not oversubscribe any input or output. Most of the commercial switches and routers are based on an input queued (IQ) architecture. The decision to move away from output queued (OQ) architecture is mainly due to the memory requirements needed for the packet buffers. For an $N \times N$ switch, an output-queued switch requires the buffer to run at $N + 1$ times the line-rate, whereas, an input-queued switch requires only twice the line-rate.

However, IQ switches which maintain a single first-in-first-out (FIFO) queue experience head-of-line (HOL) blocking and it is well known that this problem can limit the switch throughput to 58.6% even for uniform Bernoulli iid traffic[7]. It has been shown that, through the rearrangement of queues a switch can achieve higher throughput. This simple scheme is called virtual output queuing (VOQ) in which each input maintains a separate queue for each output. With this VOQ structure and the use of a maximum weight matching (MWM) algorithm [1, 2, 3] we can achieve 100% throughput, even for non-uniform traffic. However, because of high time complexity and lack of simple hardware implementation, these maximum weight matching algorithms are not attractive for practical purposes.

In the industry, a set of algorithms called "Maximal Matching Algorithms" are used. These algorithms have a low time complexity and are easy to implement in hardware [4, 5]. Some of these maximal matching algorithms achieve 100% throughput for uniform traffic. But they don't provide any throughput guarantees for non-uniform traffic. By using a moderate speedup[1] of 2, it is shown [1] that any maximal matching algorithm would deliver 100% throughput even for non-uniform traffic. Further studies [8, 9, 10] even show that with additional speedup, they can exactly emulate an output-queued switch. However, many of the results of these papers require a speedup of 2 or more.

However, from an implementation point of view, the buffering stage is still too fast for commercial DRAM in order to transfer packets at a speedup of 2. In fact, many current buffer designs only allow a speedup of 1.6 for 10 Gbps line rates. Hence, it is interesting to know if there exists a practical maximal matching algorithm that can deliver 100% throughput at a speedup less than 2 for non-uniform admissible traffic. In particular, we have analyzed an algorithm called "Fixed Priority Wrapped Wave Front Arbitration (FPWWFA)" and we have observed that the worst case throughput delivered is bounded by 3/4 or 75%. This implies that a speedup of 4/3 would be sufficient to achieve 100% throughput with this scheduling algorithm. However, it should be noted that this algorithm is not fair across different inputs and hence packets arriving on different inputs may experience different latencies.

In the next few sections, we will state our definition of speedup and describe the algorithm in detail and show the proof for the worst case throughput. The terminology of *friends*, *enemies* and *strangers* is introduced and a formula for calculating the exact service rate for the lowest priority flow in a 3×3 switch is provided. We will also show simulation results that match this formula.

3 Background

All our discussion would be based upon the switch model shown in Figure 1. Although packets arriving to the switch may be of variable length, we will assume that they are treated internally as fixed length "cells". This is common practice in high performance switches where variable length packets are segmented into cells as they arrive, car-

[1] A switch with a speedup of s can transfer upto s packets from each input and deliver up to s packets to each output within a time slot.

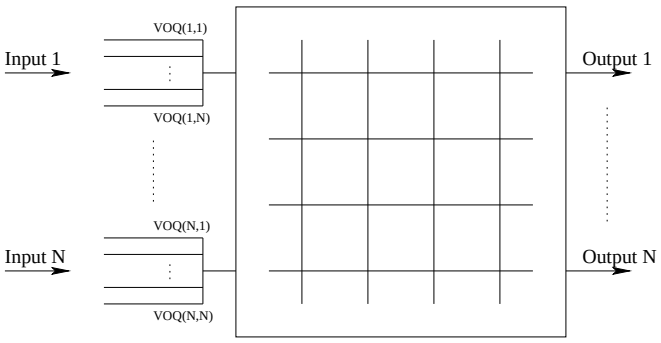

Figure 1: Switch Model

ried across the switch as cells, and reassembled back into packets before they depart. We take the arrival time between cells as the basic time unit. So, the rate of arrivals to any queue should be interpreted as the rate of fixed length cell arrivals. We represent the input rates using the notation $\lambda_{i,j}$, where *i* indicates the input and *j* indicates the output. For admissible traffic, $\sum_i \lambda_{i,j} < 1 \; \forall j$ and $\sum_j \lambda_{i,j} < 1 \; \forall i$.

Referring to Figure 1 again, all input and output buffers are assumed to have infinite capacity. Each input maintains a separate FIFO queue for cells destined for each output. Hence, there are *N* FIFO queues at each input and these are called *Virtual Output Queues (VOQs)* - with the understanding that $VOQ_{i,j}$ buffers cells at input *i* destined for output *j*.

A scheduling algorithm selects a matching between inputs and outputs in such a way that each non-empty input is matched with at most one output and, conversely, each output is matched with at most one input. The matching is used to configure the switch before cells are transferred from input side to output side. In our model, the scheduler makes one of these matchings once every time unit. So, in order to achieve stability we limit the admissible traffic and believe the inverse of this gives the speedup required. For example, if a switch can only sustain a 50% combined traffic rate, this implies a speedup of 2 is required.

Goal: Given the above switch model, determine the maximum value of λ_{max} satisfying the following conditions, while maintaining the switch stability. The traffic in assumed to be bernoulli.

$$\sum_i \lambda_{i,j} < \lambda_{max} \; \forall j \text{ and } \sum_j \lambda_{i,j} < \lambda_{max} \; \forall i$$

For rest of the paper, we will concentrate on a specific maximal matching algorithm called *Fixed Priority Wrapped WaveFront Arbitration* (FPWWFA) explained in the next section.

4 Algorithm

In this section we will describe the selected scheduling algorithm. The crossbar arbiter is an $N \times N$ array of arbitration cells, with one cell per crosspoint. For simplicity we have shown a 4×4 array in Figure 2. Each crosspoint in the crossbar can be configured independently. However, as mentioned before, a scheduling algorithm selects a matching between inputs and outputs with each input matched to atmost one output and each output is matched to atmost one input. So, in any of the matchings only one crosspoint in a given row or column can enabled.

For each crosspoint (i, j), in addition to the *request* $(R_{i,j})$ input and *grant* $(G_{i,j})$ output, it has two inputs, *north* $N_{i,j}$ and *west* $(W_{i,j})$, and two outputs, *south* $(S_{i,j})$ and *east* $(E_{i,j})$ as shown in Figure 3. Note that $N_{i,j} = S_{i-1,j}$ and $W_{i,j} = E_{i,j-1} \; \forall i, j$, with the exception, $N_{1,j} = S_{N,j}$ and $W_{i,1} = E_{i,N} \; \forall i, j$. The $N_{i,j}$ signal indicates that there are no *granted* requests for the cross points above (i, j). The $W_{i,j}$ indicates that there is no $granted$ requests for the crosspoints to the left. The G output is asserted if, and only if, the crosspoint is requested and both the N and the W inputs are asserted. Thus, $G_{i,j} = R_{i,j} \wedge N_{i,j} \wedge W_{i,j}$, $S_{i,j} = N_{i,j} \wedge \overline{G_{i,j}}$, and $E_{i,j} = W_{i,j} \wedge \overline{G_{i,j}}$. For the highest priority cells, all the N and W inputs, are set to 1. Consider the wrapped diagonals shown in Figure 4. The top priority is given to the cells belonging to the first wrapped diagonal. The arbitration cells reach their final configuration in a "wave front" that moves diagonally from the first wrapped diagonal to the nth wrapped diagonal. If a cell performs its operation in T time units, the outputs of cell (i, j) are stable in their final values after $[((i + j - 2) \mod n) + 1]T$ time units. Hence, the FPWWFA completes the arbitration in nT time units.

5 Proof

Before delving in to the proof, we will introduce the notion of *enemies*, *friends*, and *strangers*. Then, the proof will be

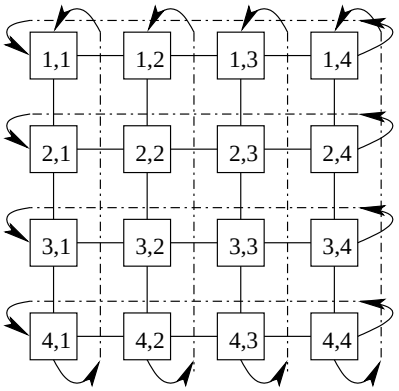

Figure 2: 4 × 4 cell array

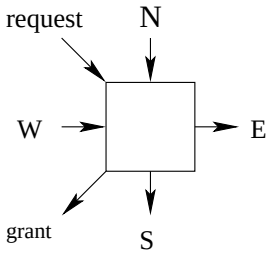

Figure 3: Crosspoint inputs and outputs

split into two parts. First part will include only the enemies and the second part will extend that to include all the cells for a 3×3 switch.

5.1 *Enemies, Friends and Strangers*

In this section we will analyze how the service rate of a given queue can be affected by other flows in the switch. Consider again the arbiter array shown in Figure 4. With the above scheduling algorithm it is clear that cells on the fourth wrapped diagonal will have the lowest priority and thus we will select the flow (4,1) as the *victim flow*. We define a victim flow as the flow that would be the first to become unstable. This point would allow us to calculate λ_{max}. Due to symmetry, any other flows on the same wrapped diagonal would be equivalent.

Consider the cell (4,1) of the arbiter belonging to the fourth wrapped diagonal. This cell will be granted only if the cells (1,1), (2,1), (3,1), (4,2), (4,3), and (4,4) are not granted. Hence, we call them the *enemies* of the flow(4,1). Although the flows (2,4), (3,3), and (3,4) don't affect the flow (4,1) directly, they do improve it's grant probability indirectly by affecting it's enemies, (2,1), (4,3), and (3,4). Hence, we call them the *friends* of flow(4,1). The remaining flows - (1,2), (1,3), (1,4) (2,2), (2,3), and (3,2) do not affect the flow (4,1), neither directly nor indirectly. So, we call them *strangers*. The above classification can be extended to a N×N switch in a similar way.

5.2 *How bad are the enemies?*

Before we analyze how the service rate of the victim flow is affected by all other flows, we would like to see how, just the enemies affect the service rate of the victim flow. The following is the traffic pattern that we are analyzing in this part and we call this the *L-pattern*. All other flows except the enemies of the victim flow have zero input traffic rate.

$$\begin{bmatrix} C_{1,1} & 0 & \dots & 0 \\ C_{2,1} & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ C_{N-1,1} & 0 & \dots & 0 \\ X & R_{N,2} & \dots & R_{N,N} \end{bmatrix}$$

For the above L-pattern, X's service rate = (1-C)*(1-R) where, C = $\sum_{i-1}^{N-1} C_{i,1}$ and R = $\sum_{j=2}^{N} R_{N.j}$. So, we get the value of λ_{max} for the *L-pattern* as 0.75, by minimizing the following function.

$$\begin{aligned} \lambda(C,R) &= (1-C)\times(1-R) + max(C,R)\,, \\ & \quad 0 \le C \le 1,\ 0 \le R \le 1\,. \end{aligned}$$

This proves that for the *L-pattern*, the selected algorithm is *weakly stable* when λ_{max} = 0.75. The remaining part of this section shows that for the *L-pattern*, the selected algorithm is in fact *strongly stable* when λ_{max} = 0.75.

5.2.1 *Is the algorithm strongly stable?*

Let $R_t \in \{0,1\}$ be r.v representing whether the *row queue* (combined queues from $R_{N,2}$ to $R_{N,N}$) at time T=t is empty or not. Let C_j be the r.v representing the time it takes for the *column queue* (combined queues from $C_{1,1}$ to $C_{N-1,1}$) to get empty $j+1^{st}$ time, since it got empty j^{th} time, $j \in \mathbb{Z}$. The arrival processes are independent (assume bernoulli) and the algorithm takes independent decisions for row and column queues.

Let S_i represent the i^{th} inter service time for the victim flow. We would like to show $E[S_i^2] < \infty, \forall i$. For the victim flow to be serviced, both (a) column queue should be empty and (b) row queue has no arrival. Each time (a) occurs, effectively X_t is decided randomly and let $P(R_t = 1) = p$. Then, the number of $C's$ taken before the victim flow could be serviced is distributed with parameter p. That is,

$S_i = C_1 + ... + C_G$, where each of C_i is i.i.d. and $G\ Geom(p)$. Also, let $c = E[C_i]$.

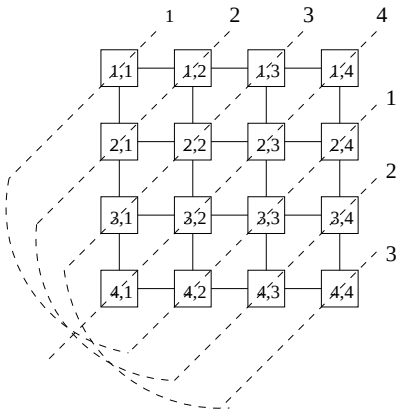

Figure 4: Wrapped diagonals in a 4 × 4 cell array

$$\begin{aligned} E[S_i^2] &= E[E[(C_1 + \cdots + C_G)^2 | G]] \\ &= \sum_{k=1}^{\infty} (1-p)^{k-1} p \, E[(C_1 + \cdots + C_k)^2] \\ &= \sum_{k=1}^{\infty} (1-p)^{k-1} p \, (kE[C^2] + (k^2-k)c^2) \\ &< \infty \end{aligned}$$

and the last equation is true, as long as $c < \infty$, and $E[C^2] < \infty$. These conditions are true because the column queue is strongly stable. Note that $E[C^2]$ is the second moment of the length of column queue's busy cycle.

5.3 *Have we seen the worst case already?*

Now we extend the above proof to include even the *friends* and *strangers*. Clearly, including strangers will not affect the final result. Regarding the *friends*, the question to ask is - *Are the friends really friends?*.

Although we haven't been yet able to prove this formally, we have some simulation results which indicate that adding friends always increases the service rate. So, we believe that *friends are indeed friends*. This means that we have already seen the worst case while dealing with just the enemies. So, we conjecture that the maximum value of λ_{max} for which the switch remains stable is 3/4.

In the next section, we provide the intuition to generating the service rate of the lowest priority flow in a 3×3 switch.

6 Throughput formula

We present a throughput formula for the flow (3,1), which is a victim flow in a 3×3 switch. Here we only consider the enemies and friends, who affect the service rate. The following is the traffic matrix we are trying to analyze,

$$\begin{bmatrix} \lambda_{1,1} & 0 & 0 \\ \lambda_{2,1} & 0 & \lambda_{2,3} \\ \lambda_{3,1} & \lambda_{3,2} & \lambda_{3,3} \end{bmatrix}$$

Queue(3,1) can only be served when $\lambda_{1,1}$, $\lambda_{2,1}$, $\lambda_{3,2}$, and $\lambda_{3,3}$ are not served. Notice that the highest priority flows are (1,1), (2,3) and (3,2). Since these flows will never be queued, let us first consider how these flows can effect the flow(3,1).

Whenever $\lambda_{1,1}$ or $\lambda_{3,2}$ arrive, we know that $\lambda_{3,1}$ is not served. So only when $\lambda_{1,1}$ and $\lambda_{3,2}$ does not arrive, there is a possibility that queue (3,1) can be served. If $\lambda_{1,1}$ does not arrive and $\lambda_{3,2}$ does not arrive, and $\lambda_{2,3}$ arrives, we know queue(3,1) can served. This is because $\lambda_{2,3}$ is the highest priority and will prevent $\lambda_{2,1}$ and $\lambda_{3,3}$ from attacking queue (3,1).

Now, it is interesting to see how $\lambda_{2,1}$ and $\lambda_{3,3}$ effect flow(3,1) when $\lambda_{2,3}$ does not arrive. The basic intuition is that $\lambda_{2,1}$ has a service rate of (1-$\lambda_{1,1}$)(1-$\lambda_{2,3}$) and $\lambda_{3,3}$ has a service rate of (1-$\lambda_{3,2}$)(1-$\lambda_{2,3}$). Thus, when $\lambda_{1,1}$, $\lambda_{3,2}$, and $\lambda_{2,3}$ do not arrive, the only time queue (3,1) is served is when both $\lambda_{2,1}$ and $\lambda_{3,3}$ are not served.

Based on the above discussion, the following is the service rate formula for queue (3,1):

$$\begin{aligned} servicerate = \; & ((1-\lambda_{1,1})(1-\lambda_{3,2})(\lambda_{2,3})) \\ & +((1-\lambda_{1,1})(1-\lambda_{3,2})(1-\lambda_{2,3}) \\ & \times(1-\tfrac{\lambda_{2,1}}{(1-\lambda_{1,1})(1-\lambda_{2,3})}) \\ & \times(1-\tfrac{\lambda_{3,3}}{(1-\lambda_{2,3})(1-\lambda_{3,2})})) \end{aligned}$$

For a sanity check, plugging in $\lambda_{2,3}$ = 0 into the above service rate gives (1-$\lambda_{1,1}$-$\lambda_{2,1}$)×(1-$\lambda_{3,2}$-$\lambda_{3,3}$).

So, in order to prove that the friends only improve the service rate, we need to show that the derivative of the service rate w.r.t $\lambda_{2,3}$ is positive.

$$\frac{d(servicerate)}{d(\lambda_{2,3})} = \frac{(\lambda_{2,1})(\lambda_{3,3})}{(1-\lambda_{2,3})^2} \geq 0$$

which is greater than zero since all the λs are between 0 and 1. This shows that friends can only help the throughput of flow (3,1), atleast in a 3×3 switch.

7 Simulations

We have confirmed the throughput formula derived in the previous section by running exhaustive simulations. These simulations once again only considered enemies and friends. The traffic rate for each individual flow was varied by increments of 0.05 where the sum of the row/column was less than 0.75. For all these traffic patterns, the system was stable.

To see how friends affect the service rate, we have run a simulation with the following traffic matrix.

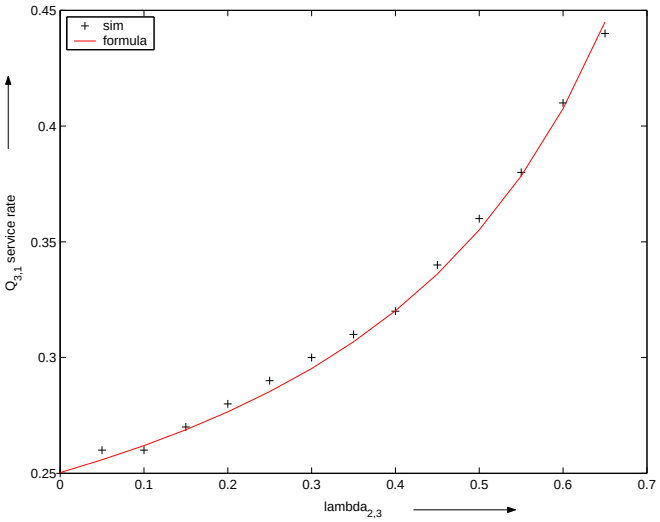

Figure 5: How friends affect the service rate?

$$\begin{bmatrix} 0.2 & 0 & 0 \\ 0.3 & 0 & \lambda_{2,3} \\ 0.5 & 0.15 & 0.35 \end{bmatrix}$$

The input traffic rate of $\lambda_{2,3}$ is varied from 0 to 0.65 in increments of 0.05. Figure 5 shows how the the victim's service rate varies with $\lambda_{2,3}$.

8 Conclusion and Future Work

As switches and line rates continue to get faster, finding solutions that fit within the memory bandwidth requirements s becoming increasingly difficult. We believe that achieving 100% throughput for non-uniform traffic has become the minimum requirement for any switch. As a result, finding the minimum speedup required to achieve 100% throughput for switches using practical maximal matching algorithms will be more important. These results will allow the switch architects to make a more informed assessment of the performance of their switch.

In this paper, we have shown that the L-traffic pattern will provide throughput guarantees with a speedup of 4/3. And through simulation, we have observed that a 3x3 switch has the same results, even when all the flows are included. We believe that, for a non-uniform Bernoulli traffic, a minimum speedup of 4/3 is required to achieve 100% throughput for a $N \times N$ switch using FPWWFA. We realize that the selected algorithm is not fair, and studies of fair maximal matching algorithms would be more interesting.

9 Acknowledgments

We would like to thank Devavrat Shah, Sundar Iyer, Isaac Keslassy and Gireesh Shrimali for helping us in our project.

References

[1] J. G. Jim Dai, Balaji Prabhakar: "The throughput of data switches with and without speedup", *Proceedings of the IEEE INFOCOM*, 2000.

[2] Nick McKeown, Venkat Anantharam, Jean Walrand: "Achieving 100% throughput in an input-queued switch", *INFOCOM '96*, pp.296-302.

[3] Nick McKeown, Adisak Mekkittikul, Venkat Anantharam, Jean Walrand: "Achieving 100% throughput in an input-queued switch", *IEEE Transactions on Communications*, Vol. 47, No. 8, pp. 1260-1267, August 1999.

[4] Yuval Tamir, Hsin-Chou Chi: "Symmetric crossbar arbiters for VLSI communication switches", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 1, pp. 13-27, 1993.

[5] Nick McKeown: "The *i*SLIP scheduling algorithm for input-queued switches", *IEEE Transactions on Networking*, Vol. 7, No. 2, pp. 188-201, April 1999.

[6] Sundar Iyer, Ramana Rao Kompella, Nick McKeown: "Designing packet buffers for router line cards", In submission to *IEEE Transactions of Networking*.

[7] M. Karol, M. Hluchyj, S. Morgan: "Input versus output queuing on a space-division switch", *IEEE Transaction on Communications*, Vol. 35, pp. 1347-1356, Dec 1987.

[8] S-T. Chuang, A. Goel,N. McKeown and B. Prabhakar: "Matching output queuing with a combined input and output queued switch", *IEEE Journal on Selected Areas in Communications*, 17, no. 6, pp 1030-1039.

[9] P. Krishna, N.Patel, A.Charny and R. Simcoe: "On the speedup required for work-conserving crossbar switches", *Presented at the* 6^{th} *IEEE/IFIP IWQOS '98*, May 1998.

[10] B. Prabhakar and N. McKeown: "On the speedup required for combined input- and output-queued switching", *Automatica*, no. 35, v.12, pp. 1909-1920, Dec 1999.