Stochastic Analysis of Stable Marriages in Combined Input Output Queued Switches

Ashish Goel [1] Balaji Prabhakar [2]

Abstract

Traditionally, Output Queued switch architectures have been proposed to implement Quality of Service schemes such as Weighted Fair Queueing. Output Queued switches with N input and output ports require up to N serial memory operations per time slot (taken to be the time between packet arrivals at an input). Given the high and increasing processor/memory gap, it is important to shift the bottleneck from memory to processor in order to obtain scalable architectures. It has recently been demonstrated that most Output Queued switches can be emulated using Combined Input Output Queued Switches which require $O(N)$ processor operations and a small, constant number of memory operations, thus moving the performance bottleneck from memory to processor. These bounds hold against all, even adversarial, traffic patterns. In this paper we analyze the scheduling algorithms used in [2, 10] to obtain the above results when the input traffic is stochastic. We prove that if the queue size at each output port in the Output Queued switch being emulated has an exponential tail, then the above algorithms need just $O(\log N)$ processor operations with high probability.

1 Introduction

A number of studies point out the need for using scheduling schemes like Weighted Fair Queueing (WFQ) and Generalized Processor Sharing (GPS) for providing Quality of Service (QoS) guarantees to flows in a packet switched network [1, 3, 8, 9]. These schemes assume an Output Queued (OQ) switch architecture, where incoming packets to a switch/router are immediately forwarded to their respective outputs. But the OQ architecture requires upto N serial memory operations per time slot (taken to be the time between packet arrivals at an input) for an $N \times N$ switch. That is the "speedup" of the memory needs to be atleast N. This makes the OQ switch architecture prohibitively expensive for high-speed implementations and for large sized switches.

On the other hand, Input Queued (IQ) switch architectures require memories to only run as fast as the line, i.e. IQ switches have a speedup of just one. This makes input queueing very appealing for switches with high line rates, or with a large number of ports. But IQ switches do not allow a fine grained control of the latency of packets, and hence cannot be used to provide strong QoS guarantees.

It has recently been shown that Combined Input and Output Queued (CIOQ) switches, along with novel scheduling algorithms, can achieve exact emulation of OQ switches at low speedups[1]. By exhibiting different switch scheduling algorithms, Prabhakar and McKeown [10] and Chuang *et al* showed that a CIOQ switch can emulate a large class of output scheduling policies at a speedup of four and two, respectively. This class includes the FIFO, strict priority, and weighted fair queueing scheduling policies. Thus, CIOQ switches can combine the advantages of OQ switches (good QoS guarantees) and of IQ switched (low speedup). The algorithms proposed in [2, 10] require at most $O(N)$ iterations[2] during each time slot.

The above results hold in the worst case – given any input (even adversarial) traffic pattern, the above algorithms exactly emulate an OQ switch without exceeding $O(N)$ processor usage. In this paper we do not propose any new algorithms; we merely analyze the above algorithms and show that their running time on stochastic input traffic is better than the worst case running time with high probability. Specifically, we show that under some weak assumptions on the in-

[1]Department of Computer Science, Stanford University. Email: agoel@cs.stanford.edu
Research supported by ARO Grants DAAG55-98-1-0170 and ASSERT award DAAG55-97-1-0221 and by ONR Grant N00014-98-1-0589.

[2]Departments of Electrical Engineering and Computer Science, Stanford University. Email: balaji@isl.stanford.edu

[1]We do not give a detailed history of the problem but mention only the results directly relevant to this paper; a detailed history can be found in [2].

[2]Both these solutions use a parallel matching algorithm that works in phases. Each phase takes a constant time, and therefore, the running time of these algorithms can be measured in terms of the number of iterations. For details , see [2, 10].

put traffic, the expected number of iterations is in fact $O(\log n)$, for the algorithm presented by Prabhakar and McKeown as well as the one by Chuang *et al* . Further the number of iterations has an exponential tail, and is $O(\log N)$ with high probability. The above results hold even if arrivals for different output ports are correlated.

Our analysis makes the tradeoff more appealing: Rather than use $O(N)$ memory operations and a small number of processor operations in the OQ switch, one can use $O(\log N)$ processor operations and a small number of memory operations. This suggests that the solutions proposed in [2, 10] are truly scalable to high speeds and to large sized switches. Preliminary simulation work corroborates our analysis, and indicates that the constant hidden by the Big-Oh notation in $O(\log N)$ bound is small for realistic arrival rates [7]. It is important to observe that we do not change the solutions proposed in [2, 10] in any way. Nor do we inhibit the ability of these solutions to emulate OQ switches under adversarial traffic. We merely study the performance of these algorithms under stochastic inputs.

Section 2 provides a brief summary of the algorithms that will be analyzed. The algorithms use a stable marriage algorithm as a subroutine. In Section 3 we prove that both these algorithms take $O(\log n)$ iterations with high probability, as long as the input traffics are such as to give *output* queue sizes with an exponential tail.

2 Background: Exact mimicking of output-queueing

The algorithms proposed in Chuang *et al* [2], and Prabhakar and McKeown [10] assign an input priority and an output priority to packets that are queued on the input side of the CIOQ switch. Having assigned these priorities, each input port ranks all the output ports in order of its preference. The preference of an output port X for an input port A is determined by the input priority of packets queued at A for X. If there is more than one such packet, then the packet with the highest input priority is chosen as the representative. If there are no such packets, the input port A does not include X in its ranking. Similarly each output port ranks the input ports.

Once these rankings are done, both algorithms compute a stable marriage of input ports with output ports. The corresponding packets are then transferred across in one parallel memory operation (i.e. without any input or output contention). In this section we first define stable marriages and indicate how they are computed. Then we briefly sketch the techniques used by [10] and [2] to assign input and output priorities to packets.

2.1 Stable marriages

Consider N men and N women, where each man specifies a (possibly incomplete) ranking of all the women according to their preference as a partner, and similarly, each woman specifies a (possibly incomplete) ranking of all the men. Thus two different individuals may specify completely different rankings.

A *marriage* is a pairing of men and women. A marriage is said to be stable if for any woman Alice and any man Bob, at least one of the following three conditions is satisfied:

1. Alice and Bob are paired to each other, or
2. Alice prefers her partner to Bob, or
3. Bob prefers his partner to Alice.

Finding stable marriages is a well-studied combinatorial problem. Gale and Shapley proved that stable marriages always exist [4]. They gave an algorithm to compute these marriages; their algorithm can be thought of as a parallel algorithm that runs in at most $M-N+1$ iterations, where M is the sum of the sizes of the preference lists of all the men. Due to the special structure of the preference lists constructed by [2, 10], the Gale Shapley Algorithm converges in N iterations. Each iteration of this algorithm has the following two steps:

1. Each unengaged man proposes to the woman he prefers most out of those who haven't rejected him yet.
2. Each woman looks at all the proposals (if any) she got in Step 1 of this iteration as well as the man she is currently engaged to, if any. Out of all these men she chooses the one she prefers most and gets engaged to him, breaking off her prior engagement, if any.

The algorithm terminates when no new proposals get generated during Step 1.

2.2 MUCFA: Emulating an OQ switch at a speedup of four

Prabhakar and McKeown [10] first considered the question of exactly mimicking an OQ switch with a CIOQ switch employing a small speedup. Using a scheduling algorithm, called the Most Urgent Cell First Algorithm (MUCFA), they showed that this exact emulation is possible at a speedup of just four, regardless of input traffic pattern and switch size. We now present a brief summary of MUCFA.

Let TL(c) denote the time to leave for packet c; MUCFA assumes that this time is known when c arrives at the switch. At time t, define the *urgency* of packet c to be TL(c) $- t$. MUCFA computes four stable matchings between inputs and outputs during each time slot for transferring packets from the input side to the output side (hence the speedup of four). During each scheduling phase, MUCFA sets both the input and output priority of each packet to be its urgency; the lower the urgency value of a packet, the more preferred it is by its input and output. For a detailed exposition of this algorithm, see [10].

2.3 CCF: Emulating most OQ switches with a speedup of two

Chuang *et al* presented a scheduling algorithm which they called CCF (Critical Cells First). Unlike MUCFA, CCF needs to compute and schedule only two stable marriages per switch cycle and hence has a speedup of two. The CCF algorithm, like MUCFA, exactly emulates most OQ switches[3]. Each input port maintains a priority queue of input packets; an arriving packet may get inserted any where into this priority queue, but it cannot alter the relative order of packets already in the queue. The input priority of a packet is determined by its position in the priority queue; a packet is more preferred than all packets behind it in the input priority queue. The output priority of a packet is determined by its time to leave; the earlier the time to leave of a packet, the higher the preference assigned to it by its output port. To complete the description of CCF, we must describe how packets are inserted into the priority queue upon arrival.

Define the Input Thread IT(c) of packet c to be the number of packets ahead of it in its input priority queue. Define its Output Cushion OC(c) to be the number of packets buffered at its output port which have an earlier time to leave than c. A newly arrived packet c gets inserted as far back into the input priority queue as possible while ensuring that IT(c) $\leq$ OC(c). We will concentrate on a variant of CCF described in [2] where a packet c is marked active if at the current time instant IT(c) $=$ OC(c), and inactive otherwise. A detailed exposition of CCF and the variant described above is presented in [2]. When we mention CCF in the rest of this paper, we mean the above variant.

3 Number of iterations for the stable marriage algorithm

In this section we analyze the number of iterations needed for the Gale Shapley Algorithm if the input traffic pattern is stochastic, and the scheduling algorithm used is either MUCFA or CCF.

At any time instant, define the dependency graph G to be a directed graph with a vertex corresponding to each active packet that is waiting on the input side of the CIOQ switch. Let a and b be two active packets waiting at the input side. The dependency graph G contains a directed edge from a to b if and only if packet b is ahead of a either in an input priority list or in an output priority list. Clearly, if two packets share either an input port or an output port, there must be an edge between them. An edge from a to b is said to be an *output edge* if b is ahead of a in an output priority list and an *input edge* otherwise[4].

Lemma 1 (Implicit in [10]) *The dependency graph resulting from MUCFA is acyclic.*

Lemma 2 (Proved in [2]) *The dependency graph resulting from CCF is acyclic.*

It is proved in [2] that if the dependency graph is acyclic then the number of iterations needed by the Gale Shapley Algorithm is at most N. We now show that the number of iterations is in fact $O(\log N)$ with high probability under some weak assumptions on the input traffic. Define the *output depth* of a node in the graph to be the maximum number of output edges on any directed path originating at that node; similarly define the *input depth* of a node to be the maximum number of input edges on any directed path originating at that node. Define the input depth of the graph to be the maximum input depth of any node, and the output depth of the graph to be the maximum output depth of any node. At any time instant, let Q_i be

[3] Define a push-in queue to be a queue where an arriving packet can get inserted anywhere into the queue, but an arrival cannot alter the relative position on any packet already in the queue. Departures have to be from the front of the queue. CCF can emulate any OQ switch that implements a push-in queueing policy. FIFO and Weighted Fair Queueing are push-in policies. In fact, all natural queueing policies known to the authors are push-in policies.

[4] Note that if b is ahead of a in both the input and output priority lists, then a is not considered during the stable marriage algorithm.

the occupancy of the queue at the i-th output port in the reference OQ switch being emulated, and let Q_{max} denote the largest of the Q_i's. Lemma 3 relates the output/input depths of the dependency graph to the number of iterations needed by the Gale Shapley Algorithm to terminate. Lemma 4 relates Q_{max} to these depths for both MUCFA and CCF.

Lemma 3 *If the dependency graph is acyclic, the number of (parallel) iterations of the Gale Shapley Algorithm is at most one more than the*

1. *input depth of the graph if the inputs do the proposing (i.e. the inputs act as men).*
2. *output depth of the graph if the outputs do the proposing (i.e. the outputs act as men).*

Proof: We will only prove the first statement as the proof of the second statement is symmetric. Assume that the inputs do the proposing. The proof is by induction, where the inductive hypothesis is stated below. We say that an input is *terminally engaged* to an output if they are currently engaged and if this engagement will eventually turn into marriage.

Induction Hypothesis: Consider any node X in the graph at input depth k. Let I and O represent its input and output ports respectively. Within $k+1$ iterations, either I has proposed to O and has been rejected, or the input port I is terminally engaged to some output port.

Base Step: $\mathbf{k = 0}$. Suppose node X is at an input depth of 0. Clearly the packet at this node is the most preferred by its input port, and therefore I will propose to O during the very first iteration. Let Y be the packet most preferred by the output port O. If $X = Y$ then X is the most preferred by both input and output, and hence is a sink (has no outgoing edges). Therefore I will get terminally engaged to O and we have established the base case. If $X \neq Y$ then there must be an output edge from X to Y. This edge can be composed with any directed path originating from Y resulting in a directed path originating at X. Since the input depth of X is zero (i.e. there are no input edges on any directed path originating at X), the input depth of Y must also be zero. Therefore the input port of Y must have also proposed to O during the very first iteration. Since Y is more preferred by O than X, O must have rejected I during the first iteration.

Inductive Step: Suppose the inductive hypothesis holds for all nodes with input depth less than k. We will now prove it for an arbitrary node X with input port I, output port O, and input depth k. If I is already terminally engaged then the inductive step holds trivially for X. Therfore we concentrate on the case where I is not terminally engaged.

We first establish the following claim.

Claim 3.1 *Suppose the inductive hypothesis holds for all packets with input depth less than k. Let X' be any packet with input depth k, input port I' and output port O'. If I' is not terminally engaged by the end of k iterations, then I' proposes to O' by the end of $k+1$ iterations.*

Proof: Any packet Y which is queued at I' and is more preferred by I' than X' must have input depth less than k (since there is a path from X' to Y having one or more input edges). Since I' is not terminally engaged, I' must have been rejected by the output port of all such Y during the first k iterations (by the inductive hypothesis). This in turn implies that I' will propose to O' sometime during the first $k+1$ iterations. ∎

We now continue with our proof of the inductive step for Lemma 3. Since we have already assumed that I is not terminally engaged, we can conclude from Claim 3.1 that I proposes to O during the first $k+1$ iterations. Consider the set, S, of packets destined for output O that O prefers to X. Since there is a directed path from X to any node Z in S, the input depth of Z is at most k. Therefore, the input port of any node Z in S must either be terminally engaged by the end of k iterations or propose to O by the end of $k+1$ iterations (from Claim 3.1). First suppose that the input port of some node Z in S has proposed to O before the end of $k+1$ iterations. Then O will reject I some time before the end of $k+1$ iterations and the inductive claim holds for packet X. But if no such Z exists, then the input port of each node in S is terminally engaged (to some output port other than O) by the end of k iterations, or has never proposed to O and will never propose to O in the future. In other words, O will never receive a proposal that it prefers to X. Therefore I will be terminally engaged to O by the end of $k+1$ iterations. This completes the inductive proof.

Let K be the input depth of the graph. The preceeding argument allows us to conclude that for any packet X either its input gets terminally engaged or has been rejected by its output by the end of $K+1$ iterations. Therefore there will be no more proposals after $K+1$ iterations and the algorithm terminates. ∎

It is worth noting that the inputs do the proposing in CCF [2], and the outputs do the proposing in MUCFA [10]. Hence the relevant quantity is input depth for CCF and output depth for MUCFA.

Lemma 4 *For CCF, the input depth of the resulting (acyclic) dependency graph is bounded by $Q_{max} - 1$. For MUCFA, the output depth of the resulting (acyclic) dependency graph is bounded by $Q_{max} - 1$.*

Proof: For CCF, recall that the input thread (IT) of packet X is the number of packets queued at the same input port and more preferred by the input port than X. Also recall that the output cushion (OC) of packet X is the number of packets queued at the output port where X wants to go, and more preferred by the output port than X (see [2] for a detailed explanation of these quantities). CCF only considers packets with IT = OC. Each time we follow an input edge, IT goes down by 1. Each time we follow an output edge, the OC either goes down or remains the same, and hence the IT goes down by either one or remains the same. Since the IT of a packet cannot be negative, the input depth of a packet is no more than its IT (in fact input depth and IT are exactly equal, since the path resulting from following only input edges will result in input depth = IT). Therefore maximum input depth of a packet is equal to the maximum output cushion of a packet, which is bounded by $Q_{max} - 1$.

For MUCFA, recall that the urgency of a packet X is the number of packets destined to the output port of X which are currently in the switch and need to leave the switch before X. MUCFA orders packets in increasing order of urgency on the input side (so that a packet with a lower urgency value is more preferred by the input). This implies that as we traverse an input edge, the urgency of a packet cannot increase. As we traverse an output edge, the urgency of the packet must go down by at least one. Therefore, the output depth of the packet can be no larger than its urgency, which in turn is bounded by $Q_{max} - 1$. ∎

Hence the number of iterations for both CCF and MUCFA are bounded by Q_{max}. So now we have to only analyze the process Q_{max}, which depends on the processes Q_i in the following fashion.

Lemma 5 *If each of the Q_i's has an exponential tail, then the random variable Q_{max} has expectation $O(\log N)$, has an exponential tail, and is $O(\log N)$ with high probability.*

Proof Sketch: Since all Q_i's have an exponential tail, there exist constants k and $\alpha > 0$ such that $\mathbf{Pr}\left[Q_i > k + x\right] \leq e^{-\alpha x}$, for all $1 \leq i \leq N$ and for all $x > 0$. Therefore

$$\begin{aligned} &\mathbf{Pr}\left[Q_{max} > k + \frac{\log N}{\alpha} + x\right] \\ &\leq \sum_i \mathbf{Pr}\left[Q_i > k + \frac{\log N}{\alpha} + x\right] \\ &\leq e^{-\alpha x} \end{aligned} \tag{1}$$

The expectation of Q_{max} is now bounded by

$$k + \frac{\log N}{\alpha} + \int_{x=0}^{x=\infty} e^{-\alpha x} dx = k + \frac{\log N}{\alpha} + 1/\alpha$$

which is $O(\log N)$ if we fix k and α. It is clear from (1) that Q_{max} has an exponential tail; the fact that Q_{max} is $O(\log N)$ with high probability also follows in a straight forward fashion. ∎

The next theorem follows from Lemmas 1, 2, 3, 4, and 5. Note that we do not require the Q_i's to be independent. Also, arrivals to the same output port at different time instants may be correlated as long as the Q_i's have an exponential tail.

Theorem 1 *The number of (parallel) iterations of the Gale Shapley Algorithm is $O(\log N)$, in expectation as well as with high probability, for both MUCFA and CCF as long as the queue sizes at all output ports in the OQ switch being emulated have an exponential tail. Further, the number of iterations has an exponential tail.*

M/D/1 queues have an exponential tail and a finite expectation as long as the mean inter-arrival time is larger than the (deterministic) service time. Hence Theorem 1 applies when the arrivals for each output port are Bernoulli i.i.d.; arrivals at different output ports may be correlated. Below we define a more general precondition that is sufficient for Theorem 1 to apply.

Let $A_i(t)$ denote the number of packets that arrive at the switch during switch cycle t and are destined to output port i. From the theory of large deviations, it follows that one sufficient condition for each Q_i to have an exponential tail is that for each i, the variables $\{A_i(t)\}_{-\infty<t<\infty}$ be i.i.d., have expectation bounded below one, and have an exponential tail. Again, $A_i(t)$ and $A_j(t)$ need not be independent.

Acknowledgement: The authors thank Sriram Mudulodu and Srinivasan Pichai for access to their results; their simulations were run on top of a simulator written by Nick McKeown. We also thank Nick McKeown for several helpful discussions.

References

[1] J. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms", *IEEE/ACM Transactions on Networking*, 5(5):675-689, Oct 1997.

[2] S. Chuang, A. Goel, B. Prabhakar, and N. McKeown, "Matching Output Queueing with a Combined Input Output Queued Switch," *To appear in IEEE Infocom '99 and Journal on Selected Areas in Communications.*

[3] A. Demers, S. Keshav and S. Shenker, "Analysis and simulation of a fair queueing algorithm", *Journal of Internetworking Research and Experience*, pp 3-26, Oct. 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[4] D. Gale, and L.S. Shapley, "College Admissions and the stability of marriage", *American Mathematical Monthly*, vol.69, pp.9-15, 1962.

[5] M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queueing on a Space Division Switch", *IEEE Trans. Comm*, vol.35, no.12, pp.1347-1356.

[6] N. McKeown, V. Anantharam, J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch", *INFOCOM '96*, pp.296-302.

[7] S. Mudulodu and S. Pichai, "Parallel stable matching algorithm for Combined Input and Output Queueing," *Stanford University Technical Note STAN-CS-TN-99-90*, August 1999.

[8] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case", *IEEE/ACM Transactions on Networking*, June 1993.

[9] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case", *IEEE/ACM Transactions on Networking*, April 1994.

[10] B. Prabhakar and N. McKeown, "On the Speedup Required for Combined Input and Output Queued Switching," *To appear in Automatica.*