

A Note on “Matching Output Queuing With Combined Input/Output-Queued Switch” Without Requiring a “Shadow” Output Queued Switch

Saleh Al-Harthi Saudi Telecom Company, Enterprise Business Unit

Abstract: “Exact emulation” of an $N \times N$ Output Queuing (OQ) switch is achievable using an $N \times N$ Combined Input Output Queuing (CIOQ) switch. The major benefit of such emulation is the reduction of the minimum required speedup of the switching fabric¹ from N for an OQ switch to just 2, independent of N , for a counterpart CIOQ switch. This is essential in realizing high-end switches that provide quality of service (QoS) guarantees. This result was first reported in [2] and [4], apparently reached independently. A special case was also reached independently using a service-curve formulation in [1]. Although we believe that this result is a significant theoretical breakthrough, the proofs in [2] and [4] hinge on the basic assumption that the CIOQ switch keeps a record of the OQ switch to be emulated, a so-called “shadow” OQ switch. As noticed in [2] and [4], this requirement renders a practical implementation of the algorithms therein to be prohibitive in today’s technologies. In this paper, we build on the formulation and proofs of [2] to show that the desirable emulation result is achievable without requiring a “shadow” switch for almost all known scheduling policies. More precisely, it is shown in this paper that a class of scheduling algorithms exists that allows a CIOQ switch to “exactly emulate” any “tag-based” scheduling policy of a counterpart OQ switch **without** requiring the CIOQ switch to keep a record of the OQ switch to be emulated and with a speedup of only 2. Fortunately, “tag-based” scheduling policies encompasses most of the known useful policies including Weighted Fair Queuing (WFQ), strict priorities, Service-Curve based policies as SCED, and others.

I. BACKGROUND AND MOTIVATIONS

In crossbar switches design, the choice of placing the queuing mechanism at the inputs or outputs of the switch has a direct and well studied effects on the performance of the switch in terms of throughput and delay of packets through the switch. It is well known that Output Queuing (OQ) switches are ideal from this performance standpoint. The main down side of an $N \times N$ OQ switch is the requirement on its switching-fabric to have a minimum “speedup”² of N compared to a “speedup” of 1 for a coun-

terpart $N \times N$ Input Queuing (IQ) switch. The requirement of a speedup not less than N makes building high-end OQ switches prohibitively expensive and in some cases (of high link speeds and large N) impossible using today’s technologies. Motivated to solve this problem, a body of research suggested that a Combined Input Output Queuing (CIOQ) architecture may deliver the ideal performance of the OQ switch while at the same time reduce the requirement of the speedup from N to just 2, i.e., independent of N . Ideally, it is desirable to “exactly emulate”³ an OQ switch by a CIOQ switch that requires no speedup (i.e., speedup of 1). It turned out, however, that no speedup less than $2 - 1/N$ is sufficient [2]. The main theoretical result was first reported in [2] and [4]⁴, apparently reached independently. A special case was also reached independently using a service-curve formulation in [1]. Unfortunately, this breakthrough result reduces the minimum required speedup at the expense of complex scheduling algorithms, making it impractical to implement such emulation using current technologies. The main limitation of these algorithms is a fundamental assumption that a CIOQ switch needs to keep a record of the emulated OQ switch, particularly the departure time of each cell in the emulated OQ switch. The record must be maintained once the cell arrived at the CIOQ switch and until it departs. This record is termed a “shadow” OQ switch in [2]. We refer the reader to [2] for further discussion of this “high-information complexity”. Clearly, finding a practical way of implementing such emulation that gets rid of the “shadow” OQ switch requirement is desirable. In this paper, one way of doing so for almost all known scheduling policies is described.

II. DOING AWAY WITH THE “SHADOW” OQ SWITCH

In this section and throughout this paper we assume the **same model and formulation of [2]** and the same setup of an $N \times N$ CIOQ switch.

Our objective is to get rid of two undesirable requirements on the emulating CIOQ switch. First, the requirement to *know* the “time to leave” of each cell according to the emulated OQ switch. Second, the requirement on

¹All results discussed in this paper pertain to crossbar switches.

²The speedup of a switch is defined as the number of times the switching fabric of the switch can be configured in a single time slot, where the time slot is the time interval between the arrival of two consecutive cells on an input port of the switch. All input ports and all output ports are assumed to have the same speed in bits per second.

³We say that a CIOQ switch *exactly emulates* a counterpart OQ switch if, when the two switches are fed the *exact traffic*, the departure time of each packet departed the CIOQ switch is identical to the departure time of the equivalent (or clone) packet departed the OQ switch.

⁴Some errors were found in the algorithms (and hence the presented proofs) of [4]. Discussion of errors and a “fix” is found (as of this writing) at <http://www.cs.cmu.edu/~istoica/IWQoS98-fix.html>.

the CIOQ switch to calculate the “Output Cushion” of each cell, which implies communications between inputs and outputs of the switch⁵. Towards this end, we have the following key observation. To state the key observation, we need first to define what we mean by a “tag-based scheduling policy”.

Definition 1: (Tag-based scheduling policy) A tag-based scheduling policy is a scheduling policy that tags arriving cells with a number or possibly multiple numbers and then queue them in a “linear” order with the least “number” (i.e., tag) being at the head-of-the-line and serve them in that order. “Linear” order means that “tags” must be an *ordered set* that can be ordered in an increasing sense. We term the tag at the head-of-the-line, the “earliest” tag, i.e., we think of the tags in their “linear” order as points of some generalized “time”.

For example, consider a “strict priority” scheduling policy of two sessions with session 1 assigned the higher priority. In a “tag-based” policy, the tags may take the form (i, k) where, $i = 1, 2$, representing the session I.D. and priority; and $k = 1, 2, 3, \dots$, representing the order of service of cells within a particular session. As a specific example, the cell with the tag $(1, 99)$ must be served before the cell with the tag $(2, 1)$ whenever they are queued in the system simultaneously.

Key Observation: For any “tag-based” scheduling policy in an OQ switch (and hence for *any* scheduling policy that can be mapped to a “tag-based” one), the cell with the smallest “time to leave” is always the cell with the earliest tag.

This key observation is the main reason we can mimic an OQ switch running any tag-based scheduling policy by a CIOQ switch without knowing the “time to leave” of each cell departing the OQ switch to be mimicked. Scheduling algorithms that achieve this are discussed next.

A Class of Algorithms: We first present a basic scheduling algorithm and then discuss some variations and improvements to it. As discussed in [2], we need an *insertion policy* of arriving cells at the input ports of the CIOQ switch. To specify the insertion policy and the matching algorithm, we need the following definitions:

Definition 2: (Input preference list) An input preference list for each input port of the CIOQ switch is a ranking of the output ports of the switch by that input port where the most preferred output port (the highest ranking) is the destination output port of the head-of-the-line cell in queue and the next most preferred output port by that input port is the next distinct destination of the next cells in the queue and so on. Note that the list may not be complete and could be empty.

⁵The “output cushions” are assumed to be calculated based on the shadow OQ switch. However, note that for the CIOQ switch to successfully emulate the OQ switch, the “output cushions” of the CIOQ switch must be smaller than or equal to their correspondent “output cushions” of the shadow OQ switch. Hence, the CIOQ switch can be used to calculate the “output cushions” instead. If the CIOQ switch is used, the insertion policy is stricter in the sense of requiring relatively smaller (or equal) “input threads” for newly arrived cells, compared to those based on the shadow OQ switch.

Definition 3: (Output preference list) Similarly, each output port maintains a ranking of the input ports of the switch, where the most preferred input port (the highest ranking) is the input port holding the cell with the earliest tag destined to this output port, and the next most preferred input port is the (distinct) input port holding the cell with the next earliest tag destined to this output port, and so on. Ties between input ports are broken by the indices of the input ports.

A simple insertion policy: The insertion policy is simply placing an arriving packet (i.e., cell) at a given input port at the head of that input port queue.

A Basic Algorithm: Tag-Based Matching (TBM): Instead of letting the output ports “propose” to the input ports trying to obtain their most urgent cells as was done in [2] and [4], we use input-driven algorithms. In the concluding remarks, we will discuss the advantage of this seemingly minor change. As in [2], we assume four *phases* (or minislots) in each time slot. The first is an arrival phase, the second is a scheduling phase, the third is the departure phase, and the fourth is another scheduling phase. Therefore, a speedup of 2 is assumed.

Phase-by-phase description of TBM:

- Step 1 Each unmatched input (with non-zero backlog) proposes to the output of the cell at the head of the preference list of this input, announcing the tag of that cell.
- Step 2 Each output receiving proposal(s), chooses and inform (i.e., grant) the input which has the cell with the earliest tag among all proposing inputs (breaking ties by inputs indices)
- Step 3 Granted inputs are matched to their granting outputs. If there is unmatched inputs (with non-zero backlog), go to Step 1, otherwise STOP (a maximal stable matching is found and the cells used in issuing the “winning” proposals are transferred).

We adopt from [2] the definitions of “time to leave” of a cell c , $TL(c)$, the Output Cushion, $OC(c)$, the Input Thread, $IT(c)$, and Slackness, $L(c)$ (reproduced here for convenience):

Definition 4: (Time to Leave, $TL(c)$) The “time to leave” for cell c is the time slot at which c will leave the shadow OQ switch.

Definition 5: (Output Cushion, $OC(c)$) The output cushion of a cell c at any time is the number of cells waiting in the output buffer at cell c output port with a smaller “time to leave” value than cell c .

Definition 6: (Input Thread, $IT(c)$) The input thread of cell c , at any time, is the number of cells ahead of cell c in its input preference list.

Definition 7: (Slackness, $L(c)$) The slackness of cell c , at any time, equals the output cushion of cell c minus its input thread, i.e., $L(c) = OC(c) - IT(c)$.

Mainly because the matching produced by TBM algorithm is “stable”, in the sense of [2], Lemma 1 of [2] (reproduced here for convenience) holds for the TBM algorithm⁶:

⁶Note that we are also assuming a “shadow” OQ switch, however,

Lemma 1: [2] The slackness L of a cell c waiting on the input side is nondecreasing from time slot to time slot.

Proof: See [2].

Based on Lemma 1 and the “key observation”, the main result follows.

Theorem 1: Regardless of the incoming traffic pattern, a CIOQ switch that uses TBM algorithm with a speedup of $S \geq 2$ exactly emulates an OQ switch running any tag-based scheduling policy on its output ports.

Proof: The proof is almost identical to that of theorem 4 of [2] except that the “time to leave” used in the proofs of [2] is not needed in light of the above key observation. Therefore, we omit the proof for the sake of brevity. The complete proof may be found in [1].

Remarks on Tag-based Policies: It remains to consider how useful are tag-based policies? It is fortunate that the set of tag-based scheduling policies contains most of the useful scheduling policies. In fact, most⁷ “monotonic”⁸ scheduling policies are either tag-based or can be easily mapped to one. For example, consider a simple round robin policy of M ($M \geq 2$) sessions. This scheduling policy is easily mapped to a “tag-based” policy as follows. The tag may take the form (i, k) , where $i = 1, 2, \dots, M$, representing the session; and $k = 1, 2, 3, \dots$, representing the order of serving cells within a session. Each newly arrived cell is tagged and placed in either a push-in queue or alternatively in a subqueue for the respective session. In either case, the tag-based policy may be defined to serve the cell with the smallest k among cells with $i = j \text{ Mod } M$, $j = 1, 2, 3, \dots$, if the cell exists. Most importantly, all deadlines-based policies such as the Service Curve based Earliest Deadline first (SCED) policy [3] and Weighted Fair Queuing (WFQ) policies are easily mapped to tag-based policies. Therefore, the set of tag-based policies encompasses almost all known useful scheduling policies.

III. IMPROVING TBM ALGORITHM PERFORMANCE

An improvement of the TBM algorithm to reduce the complexity can be achieved by modifying the insertion policy as follows.

VOQ insertion policy: Instead of keeping one queue at each input port, Virtual Output Queues (VOQs) may be used. The VOQ at input port i holding cells destined to output port j is denoted as $VOQ(i, j)$. In the earlier insertion policy, the reason for inserting a newly arrived cell at the head of its input queue was to ensure that its slackness is non-negative by ensuring that its input thread is zero. Since the slackness of a cell at the input side is defined as the *difference* between its “output cushion” and its “input thread”, it suffices to insert a newly arrived cell in a position such that its “input thread” is less than or equal to its

the CIOQ switch does not need to keep a record of this “shadow” OQ switch. It is used only for the proofs.

⁷“Weighted” round robin is one example of the exceptions.

⁸By *monotonic* scheduling policy we mean a scheduling policy that does not change the *relative* order-of-service of any two cells already enqueued (at a given output port). This means in particular, a newly arrived cell does not change the relative order of any two cells in the queue upon its arrival.

“output cushion”. Assume VOQs are used at a given input port i , *and* at the same time assume a data structure holding an “input preference list” for that input port. When a cell arrives at i destined to output port j , it will be placed at $VOQ(i, j)$. The data structure holding the “input preference list” must be updated such that the “input thread” of this newly arrived cell is less than or equal to its “output cushion”. Note that the CIOQ switch has no knowledge of the cell’s *actual* “output cushion” since we assume no record of the “shadow” OQ switch. However, the powerful property of the “tag-based” scheduling policies stated in the “key observation” replaces that knowledge. After a moment of reflection (based on the “key observation”), one can see that it suffices to insert the newly arrived cell *behind the cell in its VOQ with the largest tag earlier than the tag of the newly arrived cell*, if such a cell exists. In the next Lemma, the complete *VOQ insertion policy* is specified and proved to guarantee that the slackness of inserted cells is non-negative.

Lemma 2: (VOQ insertion policy guarantees the Slackness is non-negative) The slackness of cells inserted (in the input preference list) using the following insertion policy is guaranteed to be non-negative:

(1) Arriving cells are placed in their respective VOQs sorted by their tags such that the cell with the smallest tag is at the head of the VOQ. The “input preference list” variable is updated as follows:

(2a) If at least one cell exists in the VOQ with a tag earlier than the tag of the newly arrived cell, then modify the “input preference list” such that the newly arrived cell is stitched (i.e., inserted) behind the cell in the VOQ with the largest tag earlier than the tag of the newly arrived cell.

(2b) If the VOQ is empty or all cells in the VOQ have tags later than or equal to the tag of the newly arrived cell, then modify the “input preference list” such that the newly arrived cell is stitched (i.e., inserted) at the head of the “input preference list”.

Proof: Denote the cell, if it exists, with the largest tag earlier than the tag of the newly arrived cell as c' . Denote the newly arrived cell as c^{new} . We first consider the case when such a cell exists. The intuition behind (2a) is two fold. First, cells destined to the same output port with *earlier tags* will be served before this newly arrived cell in an OQ switch, and hence the “output cushion” of this newly arrived cell must be *at least* one larger than the “output cushion” of the cell c' . Second, the “input thread” of the newly arrived cell was made equal to the “input thread” of c' plus one, by inserting c^{new} immediately behind c' . This guarantees that the slackness of c^{new} is non-negative as long as the slackness of c' is non-negative. In other words, note that:

$$\begin{aligned} OC(c^{new}) &\geq OC(c') + 1, \text{ and} \\ IT(c^{new}) &= IT(c') + 1. \end{aligned}$$

Thus,

$$\begin{aligned} L(c^{new}) &= OC(c^{new}) - IT(c^{new}) \\ &\geq OC(c') - IT(c') \end{aligned}$$

$$= L(c').$$

Therefore, a simple induction argument shows that if the first cell that arrived for a given VOQ was inserted according to the above policy (i.e., (2b)), all subsequently arriving cells will have non-negative slackness upon arrival.

Now we consider the case when no cell with an earlier tag than the tag of the newly arrived cell exists in the VOQ of the newly arrived cell. If the newly arrived cell is inserted at the head of the “input preference list”, then its “input thread” is zero, i.e., $IT(c^{new}) = 0$, implying that its Slackness is non-negative since its “output cushion” is non-negative by definition.

When Lemma 2 is combined with Lemma 1, it follows that the Slackness of any cell is always non-negative.

VOQs at an input port reduce the number of biddings by that input port for a particular output port to one, which in turn reduces the complexity of the matching algorithm to $o(N^2)$.

IV. CONCLUDING REMARKS AND FUTURE WORK

In this paper, a class of scheduling algorithms in an $N \times N$ CIOQ switch was introduced that allows exact emulation of most known useful scheduling policies of a counterpart OQ switch without requiring a “shadow” OQ switch. The interesting fact is that this was shown to be possible with a speedup of only 2 for the CIOQ switch, independent of N .

There are many variations to the basic TBM algorithm. We briefly described one of these variations and showed that it resulted in reducing the complexity of the algorithm. A complete discussion may be found in [1].

Another aspect of the practicality of these algorithms is that although we assumed a preference list for each output port as demanded by the structures of the proofs, in practice an output port needs not to keep a preference list. This is due to the fact that each bidding (or proposing) input port is required to announce the tag of the cell that needs to be transferred. The output port when picking the smallest tag is in effect using his preference list without keeping a record of it! This is one major advantage of using an input-driven matching algorithm over output-driven one. The intuition is that information is naturally available at the inputs.

Further simplifications and improvements are possible. Another future direction we are considering is the possible applications of this class of algorithms to wireless networks.

REFERENCES

- [1] Saleh Al-Harhi, Service-Curve Based Scheduling With QoS Guarantees in Input-Queued and Combined-Input-Output-Queued Switches, Report, Department of Electrical & Computer Engineering, University of California, San Diego, January, 2001.
- [2] S-T Chuang and A Goel and N McKeown and B Prabhakar, Matching Output Queueing with a Combined Input/Output-Queued Switch, *IEEE Journal on Selected Areas of Communications*, 17(6):1030–1039, June 1999.
- [3] R. L. Cruz, Quality of service guarantees in virtual circuit switched networks, *IEEE Journal on Selected Areas of Communications*, 13(6):1048–1056, June 1995.

- [4] I. Stoica and H. Zhang, Exact emulation of an Output Queueing switch by a Combined Input Output Queueing Switch, In *Proceedings of IWQoS'98 (IEEE/IFIP)*, Napa, CA, pages 218–224, May 1998.