

# Linear Complexity Algorithms for Bandwidth Reservations and Delay Guarantees in Input-Queued Switches with no Speedup

Anthony C. Kam, Kai-Yeung Siu  
d'Arbeloff Laboratory for Information Systems and Technology  
Massachusetts Institute of Technology  
Rm. 3-449c, 77 Massachusetts Ave., Cambridge, MA 02139  
{kam,siu}@list.mit.edu

## Abstract

*We present several fast, practical linear-complexity scheduling algorithms that enable provision of various quality-of-service (QoS) guarantees in an input-queued switch with no speedup. Specifically, our algorithms provide per-virtual-circuit transmission rate and cell delay guarantees using a credit-based bandwidth reservation scheme.*

*The novelties of our algorithms derive from judicious choices of edge weights in a bipartite matching problem. The edge weights are certain functions of the amount and waiting times of queued cells and credits received by a virtual circuit. By using a linear-complexity variation of the well-known stable marriage matching algorithm, we demonstrate by simulations that the edge weights are bounded. This implies various QoS guarantees or contracts about rate and cell delays. Network management can then provide these contracts to the clients. We present several different algorithms of differing complexity and power (as measured by the usefulness of each algorithm's contract).*

*While most of this paper is devoted to the "soft" guarantees demonstrated by simulations, we also list a few "hard" theoretical guarantees for some of our algorithms (proved in our full report). As can be expected, the provable guarantees are weaker than the observed performance bounds in simulations.*

## 1. Introduction and Motivation

Traditional switches and routers usually employ *output-queueing* – when packets arrive at an input port, they are immediately transferred by a high-speed switching fabric to the correct output port. Data are then stored in output queues, and various fair queueing

policies have been considered (see [25] for a survey) to provide various quality-of-service (QoS) features such as delay, bandwidth and fairness guarantees. In such a switch, the speed of the switching fabric and output buffer memory is required to be  $N$  times the input line speed, where  $N$  is the number of inputs. As line speeds increase to the Gb/s range and as routers have more input ports, the required fabric speed becomes infeasible unless very expensive technologies are used [12, 2].

To overcome this problem, input-queued switches are being considered (e.g., [2, 14, 16, 11]) – Incoming data cells are first stored in queues at the input side, then a scheduler chooses a subset to transfer across a slower fabric to the output side, where they might be sent along an output line immediately, or queued again for further resource management. The ratio of the fabric speed to the input speed is called the "speedup." An output-queued switch essentially has a speedup of  $N$  (whereupon input queues become unnecessary), whereas an input-queued switch typically has a much lower speedup, as low as the minimum value of 1 (i.e., no speedup), making the switch more feasible and scalable in terms of current technology and cost. However, the additional input-side queueing delay makes it difficult to provide similar kinds of QoS guarantees as an output-queued switch.

This paper presents scheduling algorithms that achieve very good results with respect to two specific QoS requirements – bandwidth reservations and cell delay guarantees – in a switch with no speedup. Section 2 states our assumptions and problem model. Section 3 reviews some previous works and explains the specific contributions of this paper in that context. Section 4 presents our algorithms, simulation results and theoretical results. Due to space limitations, the readers are referred to our full report [10] for proofs. Further discussions are given in section 5.

## 2. Problem Model and Assumptions

**Timeslotted, cell-based transmissions.** We assume that data are transferred across the switch fabric in fixed sized cells, and that time is slotted so that an input can transfer one cell to an output in one timeslot.

**Crossbar switch.** The switch fabric studied here is an  $N \times N$  crossbar, characterized by its *transmission constraints* – that at any given time, any input port can only be transmitting to at most one output port, and any output port can only be receiving from at most one input port.

**No speedup.** In this paper we assume the switch has the minimum speedup of 1, i.e., the fabric speed is equal to the input speed. We also assume all input and output speeds equal 1 cell/timeslot. The motivation is that lower speedup makes the switch more feasible and scalable in terms of current technology and costs. A speedup of 1 also provides the most stringent testing condition for our algorithms in simulations.

**Per-VC queueing.** Since we intend to provide bandwidth and delay guarantees to each individual VC, we assume each VC has its own queue.

**The Abstract Problem Statement – Matchings.** The usual abstract picture of a crossbar switch depicts it as a bipartite graph  $G = (U, V, E)$ . The input ports are nodes  $U$  and output ports are nodes  $V$ , and the edges  $E$  represent possible transmissions. The crossbar transmission constraints specify that a set of transmissions can occur at the same timeslot if and only if it corresponds to a *matching* – a subset of edges  $M \subset E$  such that each node has at most one edge in  $M$ . Most scheduling algorithms, including ours, associate a priority or *weight*  $w(e)$  to the each edge  $e \in E$ ; thus most scheduling algorithms are characterized by two separate choices – (i) deciding what to use as edge weights  $w(e)$ , and (ii) computing a matching given the weighted graph  $(G, w)$ . This matching represents the cells transmitted across the fabric at a given time. The procedure is then repeated for each timeslot.

## 3. Previous Work

Most previous work utilize existing matching algorithms or simple modifications. Their contributions derive from judicious choices of edge weights, and what performance can be proved (theoretically) or demonstrated (in simulations or experiments).

**Terminology.** Let  $v$  denote a typical VC, and let  $A_v(t)$ ,  $S_v(t)$  and  $L_v(t)$  denote the number of arrivals, the number of transmitted cells, and the queue length (in the input queues) of  $v$  at time  $t$ . All three variables are in units of cells. Note that  $L_v(t+1) = L_v(t) +$

$A_v(t) - S_v(t)$ , and that  $S_v(t) = 0$  or 1 because the fabric has no speedup. Further let  $W_v(t)$  denote the waiting time or delay (so far) of the oldest input-queued cell of  $v$ , measured in units of timeslots.

In reality many VCs may have the same input-output pair, and the scheduler must keep track of each VC separately – separate  $L_v, W_v$ , etc. Conceptually, the bipartite graph becomes a *multi-graph* where there may be many edges (many VCs) going between the same two nodes (ports). However, since we will only be interested in choosing high edge weights, a simple preprocessing can trim the multi-graph into a normal graph by choosing the highest weight VC between any input-output pair. So, from now on, we will assume that each VC has a distinct input-output pair for the sake of simplicity. Given this assumption, we can write  $S_{ij}(t), L_{ij}(t)$ , etc., when we mean  $S_v(t), L_v(t)$  where  $v$  is the unique VC that goes from input  $i$  to output  $j$ .

### 3.1. Previous theoretical work with no speedup

[23] uses  $w(e_{ij}) = L_{ij}(t)$  as edge weights and chooses a matching with the maximum total weight every timeslot. It proves that for arbitrary (non-uniform) loading, the expected queue lengths  $E[L_{ij}(t)]$  are bounded, assuming i.i.d. traffic streams and that no input or output port is overbooked. Hence, this *maximum weighted matching* algorithm (with  $L_{ij}$  weights) achieves 100% throughput. This result is later independently discovered by [15]. [17] also uses a maximum weighted matching algorithm, but with  $w(e_{ij}) = W_{ij}$ . As a result, the expected waiting times (or cell delays) are provably bounded. Unfortunately, maximum weighted matching algorithms are complex and slow (typically  $O(N^3)$  complexity with large overhead [1]) and not suitable for implementation in high-speed switches.

To overcome this problem, [18] uses “port occupancies”  $w(e_{ij}) = \sum_p L_{ip} + \sum_q L_{qj}$  as edge weights together with a specialized, faster  $O(N^{2.5})$  maximum weighted matching algorithm to provably bound edge weights (and hence  $L_{ij}$ ). [22] goes one step further and shows that, with the original  $L_{ij}$  weights,  $E[L_{ij}]$  can be bounded by a large class of randomized algorithms, some of which have  $O(N^2)$  complexity or “linear complexity” – linear in the number of edges.

All of these results are based on Lyapunov stability analysis, and consequently, all the theoretically established bounds are very loose. While the algorithm of [23, 15] exhibits relatively small bounds in simulations (see [14]), the sample randomized algorithm given in [22], which is the only “linear-complexity” algorithm above, still exhibits very large bounds in simulations.

To the best of our knowledge, no linear-complexity algorithm has been shown to have small bounds in simulations and also provide some kind of theoretical guarantee.

### 3.2. Previous theoretical work with speedup

Very recently, there are several new works dealing with QoS guarantees with speedup [19, 3, 13, 21, 5]. The earliest of these, [19], provides an algorithm that, with a speedup of 4 (or more), allows an input-queued switch to exactly emulate an output-queued switch with FIFO queues, in the sense of identical output streams. [21, 5] strengthen this result in two ways: first, their algorithms require only a speedup of 2, and second, their algorithms allow emulation of a wide class of non-FIFO fair queueing disciplines that provide various QoS guarantees. [3, 13] present several new algorithms that are not emulation-based but provide QoS guarantees that are comparable to those achievable in well-known output-queueing schemes, e.g., delay bounds independent of switch size  $N$  with speedup of 6, delay bounds dependent on  $N$  with speedup of 4, and 100% throughput with speedup of 2. The edge weights used in these works are also simple functions based on  $L_{ij}, W_{ij}$ , etc., or are parameters found in the reference output-queueing model being emulated.

Unlike the results cited in the previous section which are based on maximum weighted matchings and Lyapunov analysis, the results cited in this section are based on stable marriage matchings (or variations) and combinatorial analysis. Consequently, they typically have lower complexity (many of these algorithms have linear complexity) and much tighter theoretical bounds. However, they all require speedup of 2 or more.

### 3.3. Previous simulation studies

While theoretical studies have concentrated on the goals of bounding expected queue lengths and waiting times, various simulation studies [14, 2, 9, 8, 4] have been carried out to investigate other aspects as well, such as average delay, packet loss or blocking probabilities, etc. Some of these studies also investigated the advantage of having a small speedup of about 2-5 (much smaller than  $N$ ). As in the theoretical works cited above, the scheduling algorithms used are based on matching algorithms which are not completely new, including: maximum weighted matching, maximum size (unweighted) matching, stable marriage matchings, randomized matchings, etc.

### 3.4. Contribution of this work

This paper focuses on two QoS features, *bandwidth reservations*, and *cell delay guarantees*, in an input-queued switch with no speedup. We present several fast, practical, linear-complexity scheduling algorithms that, in simulations, support large amounts of bandwidth reservation (up to 90% of switch capacity) with low delay.

In terms of theoretical guarantees, we are only able to prove a weaker result: some of our algorithms can support bandwidth reservations of up to 50% of switch capacity, with constant delay bounds. In some sense, this can be viewed as a “dual” of the previous theoretical result that 100% throughput can be guaranteed with speedup of 2 using linear-complexity maximal matching algorithms. However, our proof techniques are very different. It is not clear if the combinatorial proof techniques prevalent in speedup  $\geq 2$  scenarios can be used in a no-speedup scenario. Indeed it is not clear whether strong QoS guarantees similar to those of output-queueing can be achieved at all in an input-queued switch with no speedup. Our proofs are based on Lyapunov stability analysis (like the no-speedup scenarios of [23, 15]), a method which yields very loose bounds. However, our algorithms exhibit tight performance bounds in practice.

Like many previous works, our novelty comes from choosing suitable parameters as edge weights to achieve our specific goals of bandwidth and delay guarantees. Also like most previous works, our scheduling algorithm is a variation of an existing matching algorithm – in our case, a variation of stable marriage matching.

## 4. Bandwidth Reservations

It is not clear a-priori what it means to have bandwidth “reserved” for a VC. Two extreme cases are intuitively clear: First, a VC sending a smooth stream of cells below its guaranteed rate should experience little delay. Second, an “overloading” VC which constantly has a large backlog should have an average transmission rate at least its guarantee rate. Unfortunately, it is less clear what should happen when the VC is very bursty and sometimes transmits at a very high peak rate and sometimes becomes idle, even though its average rate is comparable to its guaranteed rate. We propose to clarify this issue by providing *contracts* with our algorithms. Each VC (or user) can understand exactly what service it can expect from our algorithms.

We envision a scheme where, at start-up time, each VC (or flow, or session) negotiates with network management about its guaranteed transmission rate  $g_v$ .

The network decides to grant or deny (or modify) the requests based on external factors such as priority, billing, etc., in addition to current congestion level. This paper does not consider how network management makes this decision; we only assume that the network will not overbook any resource. Since all input and output speeds equal 1 cell/timeslot, this means that:  $\sum_j g_{ij} \leq 1$  for each input  $i$  and  $\sum_i g_{ij} \leq 1$  for each output  $j$ . We will define the *reservation factor* of the switch as  $\alpha = \max(\max_i \sum_j g_{ij}, \max_j \sum_i g_{ij})$ , that is, the highest reserved load of all input and output ports.

#### 4.1. General principle of all our algorithms

Define the *credit* of a VC at time  $t$  by the following equation:  $C_v(t+1) = C_v(t) + g_v - S_v(t)$ . Intuitively, a VC gains (fractional) credits at a steady rate equal to its guaranteed rate  $g_v$ , and spends one credit whenever it transmits a cell. An equivalent view is that  $C_v(t) = t \times g_v - \sum_{\tau=1}^t S_v(\tau)$ , i.e., the VC's reserved transmissions (of  $t \times g_v$  cells) up to time  $t$ , minus its actual number of transmissions up to time  $t$ . All of our algorithms actually use the integral part  $\lfloor C_v(t) \rfloor$  as an approximation to the real number  $C_v(t)$ . The difference is negligible and we will use  $C_v(t)$  even when we mean  $\lfloor C_v(t) \rfloor$  from now on.

Recall that [23, 15, 17] use maximum weighted matchings with  $L_{ij}$  or  $W_{ij}$  as edge weights and show that the expected edge weights are bounded. Our algorithms are designed according to the same general principle – some edge weights are chosen, and we hope that a matching algorithm will make them bounded. Our algorithms use edge weights which are functions of  $L_{ij}, W_{ij}, C_{ij}$ , i.e.,  $w(e_{ij}) = f(L_{ij}(t), W_{ij}(t), C_{ij}(t))$ . The function  $f()$  is chosen carefully such that a *bound* for  $E[w()]$  corresponds to a precise bandwidth reservation *contract*. Moreover, the contract can be understood in practical and intuitive terms.

All our algorithms use fast stable marriage matching algorithms (explained next), and they only differ in the choice of edge weights (presented in subsequent sections in order of increasing conceptual and implementational complexity). Each algorithm computes a stable marriage matching  $M$  per timeslot, and then (incorporating an optimization of [22]) compares it to the matching  $M'$  used in the previous timeslot, and whichever one has the larger total edge weight is the one actually used in the current timeslot.

#### 4.2. Stable marriage matching algorithm

Stable marriage matchings have been studied since 1962 [6], and its application to input-queued switch

scheduling has also been considered before, e.g., [14, 19, 3, 13, 21]. In our context,  $M$  is a stable marriage matching if every edge  $\bar{e} \notin M$  is blocked by an edge  $e_M \in M$  of equal or higher weight, i.e., the two edges share a node and  $w(e_M) \geq w(\bar{e})$ . In switch scheduling terms, this means any VC not chosen to transmit ( $\bar{e}$ ) must be prevented to do so by another VC sharing its input or output port and with equal or higher priority ( $e_M$ ). A minor contribution of this paper is the following observation, proved in our full report [10]:

**Theorem 1** *Given a weighted bipartite graph  $(U, V, E, w)$  with non-negative weights, the total weight of any stable marriage matching is at least  $\frac{1}{2}$  of the total weight of a maximum weighted matching.*

This result, combined with the Lyapunov techniques of [23, 15] allows us to prove that some of our algorithms can support bandwidth reservations of up to 50% ( $\frac{1}{2}$ ) of switch capacity, with constant delay bounds.

There are various algorithms for computing stable marriage matchings, the original [6] back-tracking algorithm being the most commonly used in input-queued switch schedulers. For our simulations we designed a new, faster, one-pass greedy algorithm. This *Central Queue* algorithm starts from an empty matching  $M$ , and examines each edge in decreasing weight order. On examining an edge  $e$ , it is added to  $M$  if possible, i.e., if  $M \cup e$  is still a matching, otherwise  $e$  is discarded. The algorithm stops when  $M$  has reached its maximum possible size of  $N$  edges, or when all the edges have been examined. The Central Queue algorithm is thus a greedy algorithm with no backtracking, unlike the algorithm of [6]. Our full report [10] proves that this algorithm computes a stable marriage matching.

The complexity of both algorithms equals  $O(N^2)$ , i.e., linear in the number of edges, *once the edge weights are sorted*. In general, sorting would add a complexity of  $O(N^2 \log N)$ . However, most of the edge weights we use are “slowly changing” in the sense that they change by at most a small constant amount (usually 1) from one timeslot to the next. Our full report [10] describes a data structure, a doubly-linked list of bags, which can be used to maintain sort order of such edge weights from one timeslot to the next with a simple linear pass through the edges. Thus, most of our algorithms have linear complexity,  $O(N^2)$ .

#### 4.3. Credit-weighted algorithm for constantly backlogged traffic

The credit-weighted algorithm simply uses credits as edge weights  $w = C_v(t)$  and computes a stable marriage matching for transmission in each timeslot. This

explains the very simple algorithm completely. The algorithm’s complexity is  $O(N^2)$  since  $C_v(t)$  are “slowly changing” edge weights. The rest of this section describes its performance on constantly backlogged traffic, i.e., when all queues never become empty.

Our simulations use a 32x32 switch ( $N = 32$ ). The distribution of guaranteed rates  $g_{ij}$  is controlled by two simulation parameters – reservation factor  $\alpha$ , and maximum guaranteed rate  $g_{max}$ . Each  $g_{ij}$  ranges from 0 to  $g_{max}$  and each input and each output is approximately reserved to  $\alpha$ , i.e.,  $\sum_{i'} g_{i'j} \approx \sum_{j'} g_{ij'} \approx \alpha$ , for a total switch reservation of  $\approx \alpha \times N$ . The readers are referred to our full report [10] for the exact method used to generate guaranteed rates. We observe that, using our method, a few  $g_{ij} \approx g_{max}$  and many other  $g_{ij} \approx 0$ , so the reservation pattern is highly non-uniform.

As a design choice, our simulator does not allow a VC to transmit if its credit is zero (i.e., zero-weight edges are dropped from the matching), even if some switch fabric bandwidth are not used as a result. In other words, the simulated algorithms are not “work-conserving.” We make this “wasteful” choice in our simulator for two reasons: First, this represents a *more stringent* test on our algorithms – if they perform well in this scenario, they must perform even better in the non-wasteful scenario. Second, letting zero-credit VCs transmit amounts to letting them use unreserved bandwidth. We consider the sharing of unreserved bandwidth as a fairness issue which deserves a more careful treatment, and we address it in our full report [10].

Nevertheless, it is reasonable to ask whether our algorithms can exhibit high total throughput if they were “work-conserving.” The answer is yes – when zero-credit VCs are allowed to transmit, all our algorithms exhibit at least 92% and usually 97-100% total throughput. However, the rest of this paper will use the “wasteful” versions of our algorithms for a more stringent test condition.

The simulation results are shown in table 1 for various values of  $g_{max}$  and  $\alpha$ . For each different choice of simulation parameters we run the experiment 10 times and report overall upperbound figures. The quantity of interest is  $C_{max}$ , the maximum  $C_v(t)$  value achieved during the simulation, for any VC  $v$ , and for any timeslot (our simulations run for 100000 timeslots). This value can be practically treated as a “soft” bound for  $C_v(t)$  and gives rise to the following contract:

**Credit-weighted algorithm Contract:** At any time  $t$ , any VC  $v$  will have its credit bounded  $C_v(t) \leq C_{max}$ . In other words, the VC’s total number of transmissions ( $\sum_{\tau=1}^t S_v(\tau)$ ) lags behind its reserved share ( $t \times g_v$ ) by at most  $C_{max}$  cells.

$g_{max}$	$\alpha$	$C_{max}$	$LC_{max}$
0.6	90%	3	3
0.6	80%	3	3
0.6	50%	2	2
0.2	90%	3	3
0.2	80%	3	3
0.2	50%	2	2

(a) Constantly backlogged traffic

Traffic type	$g_{max}$	$\alpha$	$C_{max}$	$LC_{max}$
Bernoulli	0.6	90%	338	142
Bernoulli	0.2	90%	320	32
2-state	0.6	90%	641	253
2-state	0.2	90%	398	45

(b) Bursty, non-backlogged traffic

**Table 1. Credit-weighted algorithm**

In particular, this contract implies that the time-average transmission rate equals the guaranteed rate, as  $t \rightarrow \infty$ . The usefulness of the contract depends entirely on the bound  $C_{max}$  – the smaller (tighter) the bound, the stronger and more useful the contract. The practicality of the credit-weighted algorithm (for backlogged traffic) derives from the fact that the bounds are *very small* constants. A theoretical guarantee is offered in our full report [10]:

**Theorem 2** *If  $\alpha < \frac{1}{2}$  and all VCs are constantly backlogged, then the credit-weighted algorithm guarantees that there exists a bound  $C_{max}$  such that, at any time  $t$ , for any VC  $v$ , the VC’s credit  $C_v(t) \leq C_{max}$ .*

In other words, this algorithm is theoretically guaranteed to support any reservation pattern that does not load any input or output to more than 50%. Unfortunately, the theoretically provable bound is very loose compared to typically observed  $C_{max}$  values. Thus, the theory is much weaker than the observed performance, which exhibits small bounds even at  $\alpha = 90\%$  switch reservation. This discrepancy is most likely due to the inherent “looseness” of the Lyapunov proof technique used, and the unavailability of combinatorial proof techniques for the no-speedup scenario.

**Important footnote to any QoS contract – soft versus hard bounds:** The bound  $C_{max}$  is obtained by simulation, and is not a theoretical bound. One may have reservations about using such a bound in a “contract” or for VC admission control. However, for no-speedup scenarios, Lyapunov analysis often yields loose bounds and no useful combinatorial proof technique is known yet. Moreover, the previous works which use Lyapunov analysis in no-speedup

scenarios ([23, 22, 15, 17, 18]) only bound *expected* values of queue lengths, waiting times, etc., which are not hard bounds either. Therefore, a soft bound obtained by simulations can be considered good enough for practical purposes, especially if the VC/user recognizes the bound is obtained by simulations. Also, in today’s networks there is a large proportion of legacy, best-effort traffic that requires no bandwidth reservations. Therefore a reservation factor of 50% might not be an unrealistic assumption. In that case “stability” in the sense of bounded edge weights is guaranteed by theory, and the fact that observed bounds are much smaller than the theoretical bounds can be considered a fortunate bonus.

#### 4.4. Credit-weighted algorithm for bursty, non-backlogged traffic

We now consider non-backlogged traffic, where some queues can be temporarily empty. The algorithm must be modified to ignore such idle VCs because they have nothing to transmit. This is done by giving them edge weights  $w(e_{ij}) = 0$  regardless of their actual credit.

In our simulations we use two kinds of non-backlogged traffic: Bernoulli traffic and 2-state traffic. The detailed traffic generation process is described in our full report [10]. We mention here that for both traffic types, the number of arrivals  $A_v(t)$  is always either 0 or 1; and the average arrival rate  $\lambda_v$  is exactly the guaranteed rate  $g_v$ . We choose  $\lambda_v = g_v$  for two reasons: if the average arrival rate were higher, the VC would eventually accumulate a large backlog (a situation already studied in the previous section) because cells are never transmitted without credit, whereas if the average arrival rate were lower, the reservations will be larger than the actual traffic that needs to be transmitted and the algorithm’s job is correspondingly easier. Therefore,  $\lambda_v = g_v$  represents the *most stringent test case* for non-backlogged traffic. Bernoulli traffic is memoryless while 2-state traffic is governed by a Markov chain with mean “busy” and “non-busy” periods of 5 timeslots, and probability of an arrival is doubled while in “busy” state.

In our simulations, the credit-weighted algorithm exhibits much larger (and hence much less useful)  $C_{max}$  bounds, shown also in table 1. This is because an idle VC simply collects credits as long as it stays idle, increasing  $C_v$  without limit.

Now, as long as the VC remains idle, it is ignored by the algorithm (because  $w = 0$ ) and does not actually hurt other VCs. The problem really begins when cells arrive at this VC – it will then suddenly have a much higher edge weight  $w = C_v(t)$  compared to others, and

thus it will hog its input and output ports for a long time, transmitting every timeslot until its credit drops to a lower level comparable to other VCs. Meanwhile other well-behaving VCs sharing the same input or output port will starve and their actual transmissions will lag behind their reserved share by a large amount.

Since  $C_v(t)$  can be artificially inflated by idle VCs which do not necessarily harm anyone,  $C_{max}$  is not the best measurement to gauge the extent of this hogging problem. A better way to quantify this effect is by measuring the number of *validated cells*, defined as  $LC_v(t) = \min(L_v(t), C_v(t))$ . Intuitively, this is the number of cells of  $v$  which have existing matching credits “earmarked” for them already. These cells are not waiting for transmission due to lack of credit – they already have credit and are waiting for transmission simply because of scheduling conflicts in the switch fabric. A large  $LC_v(t)$  value means a high degree of starvation. Table 1 indeed shows large  $LC_{max}$  bounds, indicating that hogging and starvation are significant and the credit-weighted algorithm is not suitable for non-backlogged traffic. The remaining sections propose alternative algorithms that are more suitable.

#### 4.5. Bucket-credit-weighted algorithm

The bucket-credit-weighted algorithm solves the hogging problem by explicitly forbidding idle VCs to collect credits without limit. More precisely, each VC when negotiating for its guaranteed rate  $g_v$  is also given a *credit bucket size*  $B_v$ . Whenever the VC has an empty queue, if its credit exceeds its bucket size, it no longer receives any credits. In other words, if  $C_v(t) > B_v$  and  $L_v(t) = 0$  then  $C_v(t+1) = C_v(t) - S_v(t)$ , otherwise  $C_v(t+1) = C_v(t) + g_v - S_v(t)$  as before. Note that VCs with non-empty queues still receive  $g_v$  credits as before even if that would exceed its credit bucket size. After all, if a VC is busy and yet its credit exceeds its bucket size, that probably shows that the scheduling algorithm has not been serving this VC efficiently! Such a VC must not be further penalized by not receiving credits. (The idea of credit buckets have been used in other settings before, e.g., [24, 26]. However, note that our bucket sizes only restrict idle VCs.)

For simplicity, in our simulations every VC has the same bucket size. The algorithm obviously does not require this and indeed, we envision both  $g_v$  and  $B_v$  to be negotiable parameters during VC start-up – if a VC can negotiate a larger  $B_v$ , the scheduling algorithm will tolerate a higher degree of burstiness from this VC.

Table 2 shows our simulation results. The small bounds  $C_{max}, LC_{max}$  lead to a useful contract:

$g_{max}$	$B_v$	$C_{max}$	$LC_{max}$	$M_{max}$	$LCM_{max}$
0.6	40	40	38	230	170
0.6	10	10	10	305	170
0.2	40	40	18	210	135
0.2	10	10	7	350	183

(a) Bernoulli traffic at  $\alpha = 90\%$ 

$g_{max}$	$B_v$	$C_{max}$	$LC_{max}$	$M_{max}$	$LCM_{max}$
0.6	40	40	38	554	350
0.6	10	10	11	505	212
0.2	40	40	19	313	150
0.2	10	10	8	403	200

(b) 2-state traffic at  $\alpha = 90\%$ **Table 2. Bucket-credit-weighted algorithm**

**Bucket-credit-weighted algorithm Contract:** At any time  $t$ , any VC  $v$  will have its credit bounded  $C_v(t) \leq C_{max}$ . In other words, its total number of transmissions lags behind its reserved share of  $t \times g_v$  by at most  $C_{max} + M_v(t)$  cells, where  $M_v(t)$  is the total number of credits forfeited (up to time  $t$ ) due to bucket size limitation while the VC is idle.

An important question is whether a VC will lose large amounts of credits, which would defeat the purpose of reserving bandwidth. One answer is that a VC does not need to worry about losing credits unless it is idle for a long time. More precisely, if the VC is busy enough that for any time  $t$ , the total number of arrivals up to  $t$  (i.e.,  $\sum_{\tau=1}^t A_v(\tau)$ ) is at least  $t \times g_v - B_v$ , then it will never lose credit due to bucket size restrictions.

Another answer comes from simulation measurements. Our simulations measure the number  $M_v(t)$  of credits a VC has forfeited due to bucket size restrictions. We also measure  $LCM_v(t) = \min(L_v(t), C_v(t) + M_v(t))$ , which is the number of validated cells the VC would have if it had an infinite bucket size (and hence would never lose credit). Table 2 shows that the bounds  $M_{max}$ ,  $LCM_{max}$  are not negligible. (They are, however not too large either, given that our simulations run for 100000 timeslots.) Thus, the bucket size has a real effect (at least for our simulated traffic types) and any VC that agrees to such a contract with a bucket size restriction must understand the implications. If the VC is known to be extremely bursty, it might need to negotiate for a better contract, one with a large bucket size or even no bucket size restriction ( $B_v = \infty$ ) or a higher than necessary reserved rate ( $g_v > \lambda_v$ ).

The bucket size restriction explicitly bounds credits for all VCs that happen to be idle at a particular time. Our full report [10] proves that the algorithm in fact

bounds credits for all VCs at all time, for *arbitrary arrival pattern* (i.e., even if some of the traffic streams are “malicious” in some sense) –

**Theorem 3** *If  $\alpha < \frac{1}{2}$ , and each VC  $v$  has a finite bucket size  $B_v$ , then the bucket-credit-weighted algorithm guarantees that there is a bound  $C_{max}$  such that, at any time  $t$ , for any VC  $v$ , both idle and busy ones, the VC’s credit  $C_v(t) \leq C_{max}$ . This result holds for arbitrary arrival pattern.*

Again, while the theory only guarantees loose bounds at  $\alpha < \frac{1}{2}$ , simulations show a much better performance of small bounds at  $\alpha = 90\%$ .

#### 4.6. Validation as Virtual Traffic Policing and A Validated Delay Bound

Validated cells were first introduced as a reasonable measurement for gauging hogging behaviors. In fact, the concept of cell validation is much more useful, and all our remaining algorithms rely heavily on it. We will now explore this concept in more details in preparation for the remaining algorithms.

Define the validation time of a cell to be the time when it obtains a matching credit, i.e., if at its arrival the VC has  $C_v(t) > L_v(t)$ , then the cell is validated immediately, whereas if the VC has  $C_v(t) \leq L_v(t)$ , then the cell will be validated only when the VC obtains a matching credit for it. Define the validated waiting time of a cell as the time it has waited since validation.

An alternative view of validation as “virtual” traffic policing ([20, 7]) is particularly helpful here. Imagine a traffic shaping/policing module situated before the switch. The traffic police keeps track of credits. Any cell must first arrive at the policing module, where it is detained until it obtains a matching credit (i.e., it becomes validated), at which point it goes on to the input queues of the switch. In this imaginary model, the arrival process at the input ports (i.e., at the switch) will be distorted and different from the arrival process at the police – in our simulations, the arrival process at the input ports (in the imaginary model) will be neither Bernoulli nor 2-state but a sort of “capped” or “truncated” version of them. Now, an input queue length in this imaginary model equals  $LC_v(t)$  in the actual switch, and a cell’s “arrival” time to the input queue (resp., waiting time spent in input queue) in the imaginary model equals the validation time (resp., validated waiting time) in the actual switch.

Given these definitions, we can prove that, with the (bucket-)credit-weighted algorithms, any (theoretical or experimental) credit bound  $C_{max}$  implies that each cell of VC  $v$  has its validated waiting time

Traffic type	$g_{max}$	$\alpha$	$C_{max}$	$LC_{max}$	$L_{max}$
Bernoulli	0.6	90%	369	13	404
Bernoulli	0.6	50%	350	4	242
Bernoulli	0.2	90%	314	7	333
2-state	0.6	90%	616	31	671
2-state	0.6	50%	736	6	619
2-state	0.2	90%	418	10	423

**Table 3. LC-weighted algorithm**

bounded (theoretically or experimentally, respectively) by  $\lceil \frac{C_{max}}{g_v} \rceil$ . This is because if a cell is not served within this time, another  $C_{max}$  credits would have arrived which, together with the cell’s matching credit, would exceed the  $C_{max}$  bound. Note that this applies to the bucket-credit-weighted algorithm as well because, as long as the cell under consideration has not been served, the queue is non-empty and so credit bucket restrictions do not apply.

#### 4.7. LC-weighted algorithm

While the bucket-credit-weighted algorithm solves the hogging problem by explicitly forbidding excess accumulation of credits by idle VCs, in this section we take a radically different approach: the LC-weighted algorithm uses the number of validated cells,  $LC_v(t)$ , as edge weights instead of credits. These edge weights are also “slowly changing” and so the complexity remains at  $O(N^2)$ . There are no bucket sizes in this scheme.<sup>1</sup>

Table 3 shows the simulation results: tight  $LC_{max}$  bounds are observed. However, what does such a bound mean in more practical, customary and intuitive terms? The main insight is that since  $LC_v = \min(L_v, C_v)$ , if  $LC_v$  is bounded that means *either*  $L_v$  or  $C_v$  is bounded (or both). These two cases have quite different interpretations, rephrased in the contract below:

**LC-weighted algorithm Contract:** At any time  $t$ , for any VC  $v$ ,

1. If the VC has a large queue ( $L_v(t) > LC_{max}$ ) then its credits must be bounded ( $C_v(t) \leq LC_{max}$ ), i.e., its transmissions lag behind its reserved share of  $t \times g_v$  cells by at most  $LC_{max}$  cells. (Such a VC will be called “overloading” since  $L_v(t) > C_v(t)$ .)

<sup>1</sup>Bucket sizes can still be added to the LC-weighted algorithm since how to manage credits and what to use as edge weights are two independent issues. However, based on the nice simulation results for the LC-weighted algorithm without buckets, the utility of adding buckets seems doubtful.

2. On the other hand, if the VC has a lot of credits ( $C_v(t) > LC_{max}$ ) then its queue length is bounded ( $L_v(t) \leq LC_{max}$ ). (Such a VC will be called “underloading” since  $L_v(t) < C_v(t)$ .)

Table 3 also lists the maximum queue size  $L_{max}$  and maximum credit size  $C_{max}$ . Even though  $L_{max}$  is relatively large, scenario 1 above implies these must come from “overloading” VCs that have few unspent credits and are already transmitting at full reserved speed. Also, even though  $C_{max}$  is relatively large, these must come from “underloading” VCs with very short queues by scenario 2. Note that in the original credit-weighted algorithm, such “underloading” VCs are the ones that accumulate credits without limit, then hog input/output ports when they become busy again. In the LC-weighted algorithm, they still have small edge weights and do not cause any problem at all.

**A starvation/delay problem:** As defined, the simple LC-algorithm suffers from an undesirable starvation problem – the last cell before a long idle period may suffer a long delay because, throughout the long idle period,  $L_v = 1$  and hence edge weight  $LC_v = 1$  and the VC may be blocked by other VCs with higher edge weight. Note that this starvation problem occurs in any queue length based algorithm, including [23, 22, 15]. In actual implementations, an exception mechanism that kicks in after an excessive delay might be needed.

#### 4.8. Validated-waiting-time algorithm

Just as [17] uses waiting time  $W_v(t)$  to solve the starvation problem of the queue length  $L_v(t)$ -weighted algorithm of [15], we now consider using validated waiting times to solve the starvation problem of the LC-algorithm. Specifically, define the validated waiting time  $VW_v(t)$  of a VC at time  $t$  as the validated waiting time of the oldest cell. (If the oldest cell is not yet validated, that can only mean the VC has no credit, and we define  $VW_v(t) = 0$ .) The validated-waiting-time algorithm uses  $VW_v(t)$  as edge weights. Note that this requires more bookkeeping than the credit- or LC-weighted algorithms, since we now have to keep track of the individual timestamps for each cell in the queue. Moreover, these edge weights are no longer “slowly changing” and therefore sorting is required, increasing the algorithm complexity to  $O(N^2 \log N)$ .

The simulation results are shown in table 4: tight  $VW_{max}$  bounds are observed. However, what does such a bound mean in more practical, customary and intuitive terms? The main observation is that if a cell’s validated waiting time is bounded by  $VW_{max}$ , then *either* its actual waiting time (i.e., cell delay) is bounded by

Traffic type	$g_{max}$	$\alpha$	$C_{max}$	$VW_{max}$ (slots)	$W_{max}$ (slots)
Bernoulli	0.6	90%	397	45	1830
Bernoulli	0.6	50%	322	5	1740
Bernoulli	0.2	90%	292	35	3520
2-state	0.6	90%	739	77	4750
2-state	0.6	80%	480	6	1050
2-state	0.2	90%	389	48	3330

**Table 4. Validated-waiting-time algorithm**

$VW_{max}$ , or it only becomes validated at most  $VW_{max}$  timeslots ago (or both). These two cases translate into the following contract:

**Validated-waiting-time algorithm Contract:** At any time  $t$ , for any VC  $v$ , consider the oldest validated cell  $c$ .

1. If  $c$  arrived after its matching credit, or equivalently, if at its arrival the VC has more credits than queued cells (excluding  $c$ ), then  $c$  will be validated at once and will have its actual cell delay bounded by  $VW_{max}$  timeslots. (This is an “underloading” VC.)
2. Otherwise,  $c$ ’s matching credit arrived at most  $VW_{max}$  timeslots ago. Since then, at most  $VW_{max} \times g_v$  other credits have been accrued. Because  $c$  is the oldest cell, its matching credit is the oldest unspent credit, and so  $C_v(t) \leq 1 + VW_{max} \times g_v$ , i.e., the VC’s transmissions lag behind the reserved share by at most  $1 + VW_{max} \times g_v$  cells. (This is an “overloading” VC.)

Table 4 also lists the maximum actual waiting time  $W_{max}$  and maximum credit size  $C_{max}$ . Even though  $C_{max}$  is relatively large, these must come from “underloading” VCs whose cells experience small actual delays, according to scenario 1. Also, even though  $W_{max}$  is relatively large, scenario 2 above implies these come from “overloading” VCs which have few unspent credits and are already transmitting at full reserved speed.

#### 4.9. Waiting time in units of “tolerable delay”

The validated-waiting-time algorithm bounds every VC’s validated waiting time to the same bound  $VW_{max}$ , regardless of their guaranteed rates or their tolerance for delay. In some applications, however, different VCs may have different tolerance for delay. In that case, it may be more useful to give a bound/contract of this form:

**Validated-waiting-time algorithm Contract, in units of tolerable delay:** Validate waiting time of any cell of VC  $v$  is bounded by some negotiated constant  $D_v$ .

That is, the actual bound on  $VW_v(t)$  is a *delay tolerance parameter*  $D_v$  which can be different for different VCs. The delay tolerance is yet another parameter that can be negotiated during setup – VCs with stringent delay requirement must try to obtain a small  $D_v$ . This decouples the rate guarantee from the delay guarantee (see [25] for discussions of why this might be a useful feature).

The validated-waiting-time algorithm can be easily modified for this feature: simply use  $\frac{VW_v(t)}{D_v}$  instead of  $VW_v(t)$  as edge weights. The idea is to “rescale” each VC’s validated waiting time by the VC’s own delay tolerance; e.g., among two VCs with the same waiting time, the one with a smaller delay tolerance will have a correspondingly higher edge weight, and therefore it will be given priority.

In simulations, the algorithm is still observed to bound the new edge weights to small constants. Thus a modified version of the above contract is satisfied: each VC’s validated waiting time is bounded by  $K \times D_v$  for some small constant  $K$ . The size of the bound, and hence the usefulness of the contract, depends on the relative sizes of  $g_v$ ,  $D_v$ , and the product  $g_v \times D_v$  for different VCs. The exact dependence is not completely understood yet.

One particular case of possible practical interest is when  $D_v = \frac{1}{g_v}$ , i.e., the network management does not negotiate rate and delay guarantees separately but instead mandates that slower VCs (small  $g_v$ ) must tolerate proportionally larger delay. Since credits arrive in a smooth stream, measuring waiting time in multiples of “credit periods”  $\frac{1}{g_v}$  is in some sense similar to measuring number of credits. Indeed, simulations show that the VW-weighted algorithm (in units of  $D_v = \frac{1}{g_v}$ ) and the LC-weighted algorithm perform similarly. However, the VW algorithm has an important advantage – it does not suffer from the starvation problem of the LC algorithm (or any queue length based algorithm) mentioned in section 4.7.

## 5. Conclusions and Discussions

We have described several algorithms for bandwidth reservations and delay guarantees in an input-queued switch with no speedup. All algorithms use a fast, practical Central Queue algorithm to calculate one stable marriage matching per timeslot. The algorithms differ by their choice of edge weights, which are various sim-

ple functions on queue lengths, credits and validation timestamps. Each algorithm provides a different contract depending on the implications of having bounds on that algorithm's edge weights. The practical utility of our contracts derive from the fact that observed bounds are very tight. (We also prove some loose, theoretical bounds in our full report [10].) The contracts include guaranteeing transmission rates, bounded queue lengths and/or validated delays on a per-VC basis.

The same principle of choosing appropriate edge weights and bounding them can be more widely applicable to achieve other kinds of QoS guarantees, e.g., on delay jitter or fair access. In particular, our full report [10] presents some edge weights and a modified Central Queue algorithm that can establish approximate max-min fair sharing of unreserved switch capacity, between best effort traffic and guaranteed traffic.

## References

- [1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs NJ, 1993.
- [2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High-speed switch scheduling for local-area networks. *ACM Trans. on Computer Systems*, 11(4):319–352, Nov. 1993.
- [3] A. Charny, P. Krishna, N. Patel, and R. Simcoe. Algorithms for providing bandwidth and delay guarantees in input-buffered crossbars with speedup. In *IWQoS 98*, 1998.
- [4] J.-C. Chen and T.E.Stern. Throughput analysis, optimal buffer allocation, and traffic imbalance study of a generic nonblocking packet switch. *IEEE J. Selected Areas in Communications*, 9(3):439–49, Apr. 1991.
- [5] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. Technical Report CSL-TR-98-758, Computer Science Laboratory, Stanford University, Apr. 1998.
- [6] D. Gale and L. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
- [7] L. Georgiadis, R. Guerin, V. Peris, and K. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. *IEEE/ACM Trans. on Networking*, 4(4), Aug. 1996.
- [8] A. Gupta and N. Georganas. Analysis of a packet switch with input and output buffers and speed constraints. In *Proc. IEEE INFOCOM 91, Bal Harbour FL*, pages 694–700, 1991.
- [9] I. Iliadis and W. Denzel. Performance of packet switches with input and output queueing. In *Proc. ICC 90, Atlanta GA*, pages 747–53, 1990.
- [10] A. C. Kam and K.-Y. Siu. Linear complexity algorithms for QoS support in input-queued switches with no speedup. Technical report, d'Arbeloff Laboratory for Information Systems and Technology, Massachusetts Institute of Technology, Cambridge MA, 1998.
- [11] M. Karol and M. Hluchyj. Queueing in high-performance packet-switching. *IEEE J. Selected Areas in Communications*, 6:1587–1597, Dec. 1998.
- [12] S. Keshav and R. Sharma. Issues and trends in router design. *IEEE Communications Magazine*, pages 144–151, May 1998.
- [13] P. Krishna, N. Patel, A. Charny, and R. Simcoe. On the speedup required for work-conserving crossbar switches. In *IWQoS 98*, 1998.
- [14] N. McKeown. *Scheduling Algorithms for Input-Queued Cell Switches*. PhD thesis, University of California at Berkeley, May 1995.
- [15] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proc. IEEE INFOCOM 96, San Francisco CA*, Mar. 1996.
- [16] N. McKeown, M. Izzard, A. Mekittikul, W. Ellersick, and M. Horowitz. The Tiny Tera: a packet switch core. *IEEE Micro*, 17(1):27–33, Jan. 1997.
- [17] A. Mekittikul and N. McKeown. A starvation-free algorithm for achieving 100% throughput in an input-queued switch. In *ICCCN 96*, 1996.
- [18] A. Mekittikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *Proc. IEEE INFOCOM 98, San Francisco CA*, Apr. 1998.
- [19] B. Prabhakar and N. McKeown. On the speedup required for combined input and output queued switching. Technical report, Computer Science Laboratory, Stanford University, 1997.
- [20] H. Sariowan. *A Service Curve Approach to Performance Guarantees in Integrated Service Networks*. PhD thesis, University of California, San Diego, 1996.
- [21] I. Stoica and H. Zhang. Exact emulation of an output queueing switch by a combined input output queueing switch. In *IWQoS 98*, 1998.
- [22] L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *Proc. IEEE INFOCOM 98, San Francisco CA*, Apr. 1998.
- [23] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. Automatic Control*, 37(12):1936–1948, Dec. 1992.
- [24] J. Turner. New directions in communications (or which way to the information age). *IEEE Communications Magazine*, 24:8–15, 1986.
- [25] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proc. IEEE*, 83(10):1374–96, Oct. 1995.
- [26] L. Zhang. *A New Architecture for Packet Switching Network Protocols*. PhD thesis, Massachusetts Institute of Technology, Cambridge MA, 1989.