Rate Guarantees and Overload Protection in Input-Queued Switches

Hari Balakrishnan, Srinivas Devadas, Douglas Ehlert, and Arvind
Sandburst Corporation, Andover, MA 01810, USA

***Abstract*— Despite increasing bandwidth demand and the significant research and commercial activity in large-scale Terabit routers for multi-gigabit/s links, many current switch designs do not provide adequate support for rate guarantees. In particular, designs based on the popular combined-input/output-queueing (CIOQ) paradigm have unpredictable performance despite implementing sophisticated scheduling schemes on egress links, because the crossbar arbitration between ingress and egress links is done without regard to desired rate guarantees or prevailing traffic conditions. This paper describes the design of an input-queued switch system and its associated arbitration and rate allocation algorithms that achieve both absolute rate guarantees and proportional bandwidth sharing even under overloaded or adversarial traffic. Our algorithms are simple and scalable and require a switch speedup of two to provide rate guarantees; we give the theoretical justification and report on simulation results that justify our claims. A semiconductor chipset based on variants of these algorithms for routers with an aggregate capacity of 160 Gbps with links up to 10 Gbps is now commercially available, and a second-generation chipset supporting 640 Gbps will be available soon.**

I. Introduction

This paper considers the problem of providing minimum rate guarantees and proportional bandwidth sharing for traffic aggregates in an input-queued switch. These rate guarantees must be guaranteed even in the face of denial-of-service (DoS) attacks, worm-triggered traffic floods, and traffic flash crowds to suddenly-popular services, all of which cause dramatic overload in switches. An important goal of this paper to provide such *overload protection*.

Most modern routers use a crossbar-based packet switch with an *arbitration algorithm* that dynamically selects ingress to egress connections.[1] Because a purely output-queued switch requires significant switch speedup (explained below), most popular current designs use some form of combined input/output queueing (CIOQ).

Traditional CIOQ switches have large packet buffers (often on the order of 100-250ms times the line rate) at *both* the ingress and egress line cards. The two packet buffers decouple the problems of fast switch scheduling and achieving QoS: input-queueing and fast crossbar arbitration achieve high speeds, while running weighted fair queueing (WFQ) [1] or deficit round robin (DRR) [2] at the egress link provides QoS. Unfortunately, because packet buffer sizes are on the order of several Gigabytes per multi-Gbps link, the two stages of buffering significantly increase router cost and power consumption compared to a router with only one stage of buffering [3], [4]. Worse, even with two stages of packet buffers, we show in Section III that these switches do not offer sufficient overload protection for rate guarantees when the arriving traffic overloads an egress link for extended periods of time.

This paper describes algorithms for an input-queued switch with speedup 2 that achieve our goals. The key idea is to integrate switch crossbar arbitration with a bandwidth allocation algorithm that dynamically apportions bandwidth to queues in the system. Our arbitration algorithm requires only simple maximal matchings within each time slot and is both fast and scalable. In our approach, packets only cross the switch shortly before they exit the system, allowing us to dispense with deep packet buffers and QoS scheduling at the egress. Moreover, because rate guarantees are considered *before* sending packets across the switch, adverse traffic patterns do not disrupt rate guarantees.

We have also developed a distributed bandwidth allocation algorithm that uses traffic arrival and service rate information gathered on each ingress line card to dynamically determine how much bandwidth to allocate to groups of queues spread across different ingresses, and to apportion this allocation to the individual queues. The algorithm supports flexible sharing of excess bandwidth concurrent with, but independent of, minimum rate guarantees. Our results show that by allocating bandwidth among queues in the system only once every few thousand time-slots, the system remains resilient to short-term burstiness while providing long-term guarantees, enabling us to achieve our performance goals while minimizing the cost of the hardware implementation of the algorithm.

Based on variants of the algorithms presented in this paper, we have fabricated a semiconductor chipset and developed a reference switch design supporting 16 10 Gbps links for an aggregate switch bandwidth of 160 Gbps. The chipset is commercially available. A second-generation chipset based on these algorithms, which will be available soon, supports an aggregate switch bandwidth of 640 Gbps. We present the theoretical justification for our algorithms as well as simulation results that show that rate guarantees are met even under adverse overload conditions.

II. Related Work

Numerous papers have been published over the past several years on switch scheduling, rate and delay guarantees, and output emulation (*e.g.*, [3], [5]–[9]); we only survey the most

[1] We use the terms "input" and "ingress" interchangeably; likewise, "output" and "egress".

relevant related work here. As in most switch modern systems, we use the idea of virtual output queueing (VOQ) [10], [11] with per-egress queues at each ingress to avoid head-of-line blocking. Using a fluid model, Dai and Prabhakar (and Leonardi *et al.*) recently showed that for CIOQ switches with speedup 2, any maximal matching algorithm delivers a throughput of 100% if arrivals follow the strong law of large numbers and links are not overfed [9], [12]. We use this important result in our work.

Many previous schemes to achieve bandwidth QoS (*e.g.*, WFQ) on CIOQ switches without egress-link QoS functionality require complex crossbar arbitration in each time slot, which often renders them impractical. In addition, many schemes cannot handle overloaded traffic or time-varying arrival rates, and do not provide aggregate rate guarantees to groups of queues spread across ingresses but sharing the same egress. Typical approaches include multiple rounds of round-robin with priority [13], maximum-weight matchings [11], or stable matchings [14].

Based on Anderson *et al.*'s Parallel Iterative Matching (PIM) [15], Stiliadis and Verma proposed a weighted variant called WPIM to provide bandwidth guarantees in an input buffered crossbar switch [16]. Stephens and Zhang [17] describe a distributed implementation of several packet fair queueing (PFQ) algorithms, some of which give rate guarantees but may not yield 100% throughput. McKeown's iSLIP algorithm [13] is an iterative, round-robin algorithm that can achieve 100% throughput for uniform traffic. All of these approaches work in some cases, but generally have problems in coping with traffic overload, time-varying arrivals, and large differences in weights.

Kam and Siu [14] present a credit-based scheme to obtain 100% throughput and to provide minimum-rate guarantees with a speedup of 2. Their scheme performs a stable matching in each time-slot. Because each queue must be given an explicit guarantee, the scheme cannot provide aggregate guarantees for groups of queues. They also do not discuss the nature of the guarantees when a queue's arrivals vary in time. We use their credit scheme as inspiration in developing the "coloring" process that makes our proposed "tiered" maximal matching arbitration work correctly.

Hui [18] and Anderson *et al.* [15] specify a method for rate guarantees based on the Slepian-Duguid method that requires no speedup in theory; if each queue has a guaranteed rate, then this method pre-computes a switch schedule for each time-slot, which can be used until the guaranteed rate assignments change (which might happen, for instance, when the arrival rates change). Although implemented in some commercial chipsets, this approach has some shortcomings. First, without switch speedup, it cannot guarantee 100% throughput for a mix of guaranteed and non-guaranteed traffic; second, the bound on the number of iterations of the method to produce an assignment is not small, so in practice one only gets an approximation whose properties are not clear; third, and most serious from a router scaling standpoint, the amount of state required in the arbiter is proportional to the product of the number of virtual output queues and the number of time-slots to compute the schedule over, which is often large.

Chang *et al.* describe scheduling algorithms based on a classical matrix decomposition result of Birkhoff and von Neumann [19], [20]. This approach requires knowledge of the arrival rates, and uses a complex, off-line $O(N^{4.5})$ algorithm to obtain 100% throughput without switch speedup. The main problems with this scheme are the state requirements and the need to perform a complex calculation when arrival rates change.

Koksal *et al.* develop and prove the correctness of a method called *rate quantization* to convert the set of desired rates into a certain discrete set in such a way that the complexity and the rate guarantees can be greatly improved over a Birkhoff switch [21]. Moreover, quantization enables them to develop a Slepian-Duguid-like algorithm that enables the switch to both adapt to dynamically varying traffic and simplify switch scheduling.

Priority modulation is a scheme implemented in some commercial switches. It is a distributed scheme that measures egress transmission rates to control the ingress rates in a feedback control loop. When an egress determines that an ingress has received more than its allocated rate, it sets the ingress to a low priority; otherwise, it is at a high priority. In each time slot, egresses make grants to ingresses by preferring higher priority requests to lower ones; when granted a request, an ingress can send any packet it wants. Although this scheme copes better with overload than traditional CIOQ switches, it has significant problems. It does not have any known provable properties under overload, and it is unknown if a speedup of 2 suffices to meet guarantees. It also requires more than a simple FIFO at the egress links. Finally, the time-scales over which it provides protection are longer than the per-time-slot control of our solution.

Chiussi *et al* describe a scheme for scalable matching. Their scheme does not protect against overload and assumes an admissible traffic pattern that does not overfeed any egress or ingress link. While no guarantees are given for speedup less than 2, simulations indicate that for many types of admissible traffic patterns a speedup of 1.1 suffices to obtain maximum throughput [22].

III. System Architecture

A. Model

We consider an $N \times N$ switch employing VOQ with only ingress packet buffering. Each ingress has about 100 or 200ms (a typical Internet path's round-trip time) of buffering. At each ingress, there are $C \geq 1$ queues per egress, corresponding to C different configurable classes of service. In practice C is on the order of several thousands. Each ingress line card has a lookup and classification module that classifies each packet to one of these C VOQs, and enqueues the packet on it. Each egress line card has only a FIFO buffer, with capacity for only a small number of packets relative to the ingress packet memory (a few tens of KBytes versus many hundreds of MBytes).

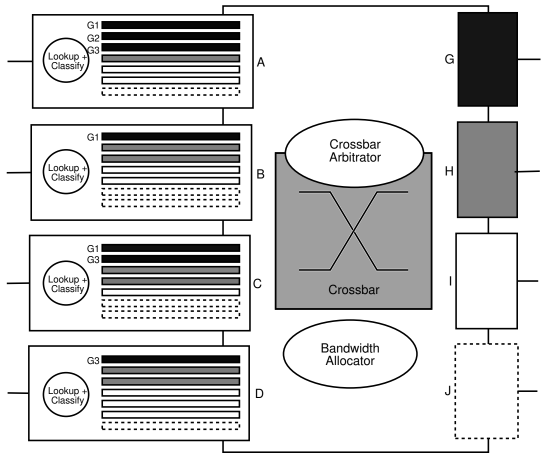

Fig. 1. Switch architecture showing a 4×4 switch. Each ingress has $C \geq 1$ VOQs. The arbiter matches ingresses to egresses in each time-slot. The bandwidth allocator periodically determines and updates the rates assigned to each queue.

Time proceeds in units of a *time-slot* of duration TS seconds. In each time-slot, each ingress is connected to at most one egress for unicast traffic, and no egress is connected to more than one ingress. The switch fabric has *speedup* S, which means that at most $(S \cdot R \cdot \mathrm{TS})$ bits can be sent across each connected ingress-egress pair in each time-slot, where R is the line-rate of each ingress and egress link in bits/second. "TSbits" is the number of bits that can arrive in one time-slot at an ingress; it is equal to $(R \cdot \mathrm{TS})$ bits. We note that for throughput guarantees this is equivalent to a speedup model where the crossbar switch computes S matchings in each time-slot with $R \cdot \mathrm{TS}$ bits being sent across after each matching. Our model may result in slightly increased delay. On the other hand, it makes for a more relaxed hardware implementation of the matching because the entire time-slot can be used to compute a matching.

We consider only switches that operate on fixed-sized packets in this paper, where each packet is TSbits in length. Our algorithms work and the results of this paper hold even for variable-sized packets, but the router requires additional machinery to handle packet fragmentation.

Network operators can configure the router to provide service level agreements (SLAs) for traffic aggregates. Each SLA operates on a *queue group*, defined as a set of queues in the system. Each queue belongs to exactly one queue group. The SLA takes the form of a minimum rate guarantee for the traffic aggregate defined by the corresponding queue group. If the minimum rate guarantee for queue group i is σ_i and the aggregate arrival rate to the queue group is a_i, then the router ensures that the rate r_i provided to the queue group is at least $\min(a_i, \sigma_i)$, and does not exceed a_i. Figure 1 shows an example where the ingresses A through D each have queues destined for egress G. The figure shows three queue groups, G1, G2, and G3, and their constituent queues.

Defining SLAs in terms of queue groups rather than individual queues facilitates flexible arrangements of queues into groups. The simplest queue group is a single queue. Another useful example is when an egress link's bandwidth is divided amongst queue groups whose constituent queues belong to different ingresses. We constrain all the queues within a queue group to share the same egress link. This is not the most general definition, since this constraint disallows a queue group from having queues destined for different egress links. However, it still permits a large number of practical SLAs.

B. Motivation

Our approach is motivated by the observation that traditional CIOQ switches with speedup 2 do not provide adequate rate guarantees when egresses or ingresses are overfed. When overfed, packets may end up going across the crossbar for queues that have already received more than their guarantees, even when there are packets on other queues that have not yet received their guarantees. This problem is not easily solvable using simple round-robin strategies for the crossbar—what is needed is a crossbar scheduling scheme that uses information from the ingress queues in making matching decisions in each time-slot.

For example, consider a switch with speedup 2 where three ingress links with arrivals at 1 Gbps all have packets destined for a 1 Gbps egress link. Suppose our goal is to guarantee rates of 0.8 Gbps, 0.1 Gbps and 0.1 Gbps to these ingress link queues. In the traditional CIOQ+egress-link-scheduling design, even assuming the crossbar does round-robin scheduling, each of these queues gets 0.67 Gbps of its traffic to the egress, since the crossbar has speedup 2 and the aggregate traffic arrival into the egress link can be 2 Gbps. At this stage, it is too late to ensure the specified rate guarantees, because only 0.67 Gbps instead of 0.8 Gbps was allowed for the first queue.

Allocating the crossbar in a weighted round-robin fashion with weights proportional to the rate guarantees does not work when packet arrivals are variable. In general, we can make the above example much worse, especially as switch sizes scale up.

C. Tiered Maximal Matching

Rather than schedule the crossbar without regard to bandwidth requirements, our scheduling scheme pays attention to whether queues have received their allocated rates or not. We are able to do this scheduling using a simple *two-tier* maximal matching strategy. The idea is to have the ingress line cards inform the arbiter of whether the allocated rates for each queue have not yet been met, in which case the queue is *hungry*, or if they have, in which case the queue is *satisfied*. The arbiter uses this information to produce a conflict-free matching of ingresses to egresses in each time-slot. It first does a maximal matching on the hungry queues, and then moves to the satisfied ones. We call this the *HSA (hungry-satisfied arbitration)* algorithm. This scheduling scheme is work-conserving and ensures 100% throughput across the system, even if not all the bandwidth of an egress link has been guaranteed.

To perform HSA, ingress line cards need to do more than in traditional CIOQ systems. They implement a credit-based scheme called *CCU (conditional credit update)* to decide if a queue is hungry or satisfied. In addition, because the actual rates assigned to a queue depend on the arrival rate of packets into that queue as well as the arrivals to other queues in the queue group located at other ingresses, they monitor the arrival rate and periodically pass this information to a bandwidth allocator. The bandwidth allocator implements an algorithm called *BAA (bandwidth allocation algorithm)* that determines the rate that should be allocated to each queue such that the SLA guarantees are met and excess bandwidth is shared properly. BAA updates each queue's allocated rate once every few thousand time-slots. The algorithms described in this paper are correct as long as the allocated queue rates do not "oversell" any ingress or egress link; *i.e.*, the sum of the minimum guarantees on any ingress or egress link is not larger than the link's line rate. Any reasonable configuration satisfies this property.

D. Advantages and Limitations

Our approach has three advantages over conventional WFQ-over-CIOQ designs. First, by assigning rates to queues dynamically based on arrivals and SLA guarantees, and arbitrating for crossbar bandwidth using HSA, our approach provides rate guarantees and bandwidth isolation even under overload when packet arrivals are not doubly substochastic. The tiered maximal matching required in HSA is simple and can therefore be performed every time-slot (*e.g.*, a few hundred nanoseconds) even for large switches.

Second, our approach enables a more flexible degree of bandwidth sharing than permitted by traditional CIOQ+WFQ/DRR systems because it separates minimum guarantees from excess sharing.

Third, our approach tightly integrates traffic management and QoS scheduling, and requires only one stage of packet buffering; the egress is a simple FIFO with space only for a small number of packets.

Like conventional WFQ-over-CIOQ designs, our approach does not give provable delay guarantees; however, in practice simulations show good delay properties for our approach because we use a token bucket scheme to determine if a queue has obtained its guaranteed rate or not.

Finally, we note that any matching scheme can be used in conjunction with the credit update and packet coloring algorithms described in Section IV. Using these algorithms, any maximal matching scheme with a crossbar speedup of 2 provides 100% throughput and rate guarantees, even under traffic overload. Lower speedups may suffice to obtain rate guarantees and 100% throughput for many types of overloaded or doubly stochastic arrival patterns.

IV. HSA Algorithm

Each ingress line card keeps track of whether each of its queues is "hungry" or "satisfied"; informally, a hungry queue is one that has not yet received its allocated rate g, while a satisfied queue is one that has received its minimum rate.

We define a *backlogged* queue to be one that currently has at least $S \cdot \text{TSbits}$ bits enqueued, where S is the speedup of the switch. A queue that has no data in it is called *empty*.

In each time-slot, each ingress provides the following information to the crossbar arbiter:

1) A list of egress links for which *some* queue at the ingress is hungry and backlogged.
2) A list of egress links for which *some* queue at the ingress is satisfied and backlogged.

For the moment, ignore the details of how the ingress determines whether a queue is hungry or satisfied; for now, imagine a scheme where the ingress monitors the service rate for each queue and compares it with the allocated rate g for the queue.

At any given time, the FIFO buffer at an egress link may be full and may not have enough space to accommodate the bits that come across the crossbar from an ingress. If there is no space at an egress, then no ingress should be matched to that egress. To prevent this matching from happening, in each time-slot, each egress informs the arbiter whether it is *eligible*, *i.e.*, whether it has enough buffer space available to accept incoming data from an ingress.

In each time-slot the arbiter knows, for each ingress, the list of hungry and satisfied ingress-egress pairs from among the eligible egresses. The arbiter uses this information to produce a maximal matching of ingress to egress links. The maximal matching produced by HSA is not an arbitrary one; indeed, an arbitrary maximal matching will *not* in general allow the system to meet the rate guarantees for the queues. HSA first produces a maximal matching by considering only the hungry ingress-egress pairs. Then, from among the unmatched ingresses and egresses (if there are any remaining), it produces a maximal matching from the satisfied queues. The result is a *tiered* maximal matching of the crossbar demand matrix, with a satisfied ingress-egress pair matched together only if there were no hungry matches for that ingress or egress.

A. Conditional Credit Update Scheme

Each ingress line card implements a *conditional credit update (CCU)* scheme for each of its queues to decide if the queue is currently hungry or satisfied. The input to CCU is the rate g that has been assigned to this queue, based on information computed by the bandwidth allocator (Section V).

Over time, a queue generally alternates between backlogged and empty periods. Let the total amount of time spent by the queue in the i^{th} backlogged epoch be T_i and suppose the rate allocated to the queue is g. With each queue, we associate the notion of a *credit*, c. Informally, a queue's credit increments at a rate g as long as it is backlogged. Whenever s bits are sent from the queue across the crossbar, c reduces by s. A queue that meets its rate would have non-positive credit. Figure 2 gives the pseudocode for the credit update algorithm that operates on each queue.

Definition 4.1: Using the CCU procedure shown in Figure 2, if the credit value of a backlogged queue is strictly

```
PROCEDURE CCU(q_i, g, pkt)
At beginning of each time-slot:
if (pkt arrives)
    ENQUEUE(q_i, pkt)
if (length(q_i) > 0)
    c[q_i] ← c[q_i] + g·TS
return;
At end of each time-slot:
sent ← number of bits sent across crossbar
        from VOQ on queue q_e
if (c[q_e] > 0) // decrement only if queue is hungry
    c[q_e] ← c[q_e] − sent;
HUNGRY ← NULL // queues with > 0 credits
SATISFIED ← NULL // queues with ≤ 0 credits
foreach backlogged queue q to egress link e
    if (c[q] > 0)
        HUNGRY ← HUNGRY ∪ {e}
    else
        SATISFIED ← SATISFIED ∪ {e}
}
Send HUNGRY and SATISFIED to arbiter
return;
```

Fig. 2. The conditional credit update (CCU) algorithm that runs on each ingress line card. q_i is the ID of the VOQ where the packet pkt arrived and g is the rate allocated to the queue.

greater than zero, the queue is called *hungry*. A backlogged queue that is not hungry is called *satisfied*.

Definition 4.2: An $N \times N$ matrix A with elements a_{ij} is called *doubly substochastic* if

1) $\forall j \ \sum_{i=1}^{N} a_{ij} \leq 1$, and
2) $\forall i \ \sum_{j=1}^{N} a_{ij} \leq 1$.

If all the inequalities above are strict equalities, the matrix is called *doubly stochastic*.

B. HSA with CCU Meets Rate Guarantees

At any time, CCU determines whether a VOQ is hungry or satisfied. CCU can also be conceptually viewed as a token bucket filter applied to the VOQ which colors packets *hungry* or "H" depending on the credit value. We can view CCU as conceptually equivalent to the following process.

1) Conceptually partition each ingress VOQ into two smaller queues: the *hungry* "H" queue, and the *pending* "S" queue.
2) In each time-slot, prior to the arbitration decision, if the credit value for the VOQ is positive, and the pending queue is non-empty, move the packet at the head of the pending queue to the hungry queue. Then, decrement the credit value by the number of bits in the packet. (In this conceptual process, we decrement credits when packets leave the pending queue as opposed to decrementing when packets are sent across the crossbar.) Otherwise, the packet at the head of the pending queue remains there.

Packets in the hungry queue are said to be *colored "H"*. The intuition behind why the combination of HSA and CCU provides rate guarantees is that CCU may be viewed as coloring the packets it sees as they arrive, in a manner that ensures that the rates of all the "H" packets in the system across all the ingresses form a doubly substochastic matrix.

Lemma 4.3: The $N \times N$ matrix, $\mathcal{H}$, formed by the rates at which CCU applied to all the VOQs colors packets as "H", is doubly substochastic, if the rate guarantees themselves form a doubly substochastic matrix.

Proof: The rate at which CCU colors "H" packets for a VOQ is limited by the rate guaranteed to the queue, when averaged over all durations when the VOQ is backlogged. Over a large enough time duration T, a VOQ between ingress i and egress j colors no more than $g_{ij}T$ packets as "H", where g_{ij} is the VOQ's guaranteed rate. Since the matrix of g_{ij} values is doubly substochastic, so is $\mathcal{H}$. ■

Theorem 4.4 (Theorem 2 in [9]): If the arrivals to a switch with speedup 2 are doubly substochastic, then the switch can achieve 100% throughput with any maximal matching arbitration.

Corollary 4.5: If the rate guarantee matrix is doubly substochastic, CCU on the ingress queues together with the HSA algorithm for switch arbitration on an $N \times N$ switch with speedup $S \geq 2$ does not cause any "H" queue to grow unbounded, and every credit value has a finite upper bound.

To see the intuition behind this result, consider any maximal matching. Suppose the queue (i, j) on ingress i for egress j is hungry, *i.e.*, $c_{ij} > 0$. If this queue is hungry in time-slot t, then HSA's tiered maximal matching ensures that the quantity $L_{ij} = \sum_k c_{ik} + c_{kj}$ in time-slot $t+1$ cannot be larger, because either queue (i, j) sends $S{\cdot}$TSbits of data, or some other queue in row i or in column j sends this many bits of data. At the same time, the sum of the credits for row i and column j can grow by at most 2. This means that every queue either eventually becomes satisfied or becomes empty.

Dai and Prabhakar [9] use this observation about L_{ij} (for them the L is the sum of actual queue lengths with doubly substochastic arrivals, not credits) to derive a Lyapunov function $V = \sum_{ij} q_{ij} L_{ij}$. The same proof flow can be used for our definition of L_{ij} to show that our equivalent V has an upper bound, which implies that each c_{ij} has an upper bound. This means that each queue receives its guaranteed rate.

This result shows that *any* strategy for determining whether a VOQ is hungry or satisfied can be used in conjunction with HSA, as long as the rate at which the strategy colors "H" packets forms a doubly substochastic matrix across the N^2 ingress-egress pairs.

1) Remarks on HSA: An easy and useful generalization of HSA's two-tiered matching is to have multiple levels in the tiered maximal matching, where the above definitions of "hungry" and "satisfied" are just two of the tiers. With this generalization, the HSA algorithm will have as many steps as there are total number of priority levels. This will allow for delay-sensitive scheduling across the priority levels.

This generalization is different from standard approaches of matching based on priority levels, because the mapping between any queue and the "hungry" and "satisfied" levels changes with time depending on the service received by the queue.

2) Remarks on CCU: In Figure 2, a queue's credits are decremented after data is sent only when the queue is hungry before the transmission and not when it is already satisfied. The precondition to decrementing is not required to prove asymptotic rate guarantees. However, if we did not decrement conditionally, a queue's credit might come down to a large negative value, by virtue of transmissions done when it was already satisfied and when the egress link wasn't being contended for by other hungry queues. Later, if the egress link became a point of contention from other competing ingresses, the queue with large negative credits may end up being starved for long periods of time. Thus, we "forgive" transmissions done during periods of little competition, and don't hold that against a queue.

Credits are incremented only when the queue is non-empty and not otherwise. This is to avoid a queue that has had no arrivals for a long time from obtaining a large positive bank of credits, and using that later. Specifically, the times at which credits get incremented affects the time-scales over which we can claim that packets are colored "H" at a doubly substochastic rate in the system. If we ensure that this happens only when a queue is backlogged, then this double substochasticity holds at all times.

We note that the pseudocode shown in Figure 2 may add some latency to a packet that arrives on an empty queue whose credits are at $-S \cdot \text{TSbits}$, because the packet may have to wait until the credit increases to > 0 for the queue to be hungry. In that time, the queue may not get to send the packet if there are other hungry queues for the same egress elsewhere in the system. To counter this, we also allow a queue's credits to increment when the credits are negative, regardless of whether the queue is empty or not. Again, this does not affect the rate guarantee results proved in this section but improves forwarding latency.

3) Egress FIFO: The proofs of this section work when the egress link employs a simple FIFO across all queues, but do not say anything about how much buffering is required at the egress. One potential concern is whether a finite amount of egress FIFO buffering causes the results to be invalidated. We show an upper bound for the required egress FIFO size.

CCU ensures that hungry packets resemble a token bucket whose drain rate is at most σ_i and whose bucket depth is $B = S \times \text{TSbits}$ bits. Supposing, for the moment, that only hungry packets make it to an egress link that has Q queues feeding it, then a buffer of $N{\cdot}B{\cdot}Q$ bits suffices because packets also drain from the egress at a rate larger than $\sum_i \sigma_i$. In fact, it is even smaller because the crossbar constraint ensures that all bursts cannot occur simultaneously; a simple modulation scheme at an ingress link to prevent all its queues on the same egress from bursting simultaneously ensures that the buffer size needed is only $N \cdot B$ bits, which is much smaller than the ingress packet buffers that run into Gigabytes.

To achieve high utilization, we want to service satisfied queues too. Satisfied packets do not go through the CCU token bucket. Therefore, it is possible that satisfied packets that make it through fill up the FIFO and block hungry packets in succeeding time-slots. One approach to solve this problem is as follows. Given an egress FIFO of size B', only service satisfied queues if the amount of data in the FIFO is $\leq \frac{B'}{S}$, where S is the speedup. If satisfied packets fill up this space, the data will drain out in $\frac{B'}{S \cdot \text{TSbits}}$ time-slots, which is exactly the number of time-slots required for hungry packets to fill up the buffer of size B'.

V. BAA: Bandwidth Allocation Algorithm

In Section III we defined and motivated the idea of queue groups. Given an assignment of rates g to individual queues, Section IV showed how the ingress line cards and arbiter ensure that the assigned rates are met. This section presents BAA, an algorithm that assigns these g values to the individual queues.

Consider a queue group j that has been guaranteed a rate σ_j in an SLA. Our goal is to ensure that we allocate the smaller of the group's demand[2] and σ_j to the queue group, dividing the allocation to the constituent queues in proportion to their relative demands. Once bandwidth is assigned such that the SLAs are met, some egress links may have excess bandwidth available for queue groups to use (*e.g.*, because some queue groups don't have enough current demand, or because not all of the egress link's bandwidth has been guaranteed). BAA apportions the excess bandwidth on each egress link in weighted-fair fashion to groups that have residual demand, according to an *excess allocation vector*, ϕ.

BAA ensures that the resulting matrix of g values to the different queues is doubly substochastic, so that HSA can run correctly. A static assignment of g values to queues based on the SLA-guaranteed rates σ does not work because we need to both apportion bandwidth to the queues in the queue group, and also ensure that queue groups with demand less than their σ relinquish the rate difference. Thus, BAA is an essential step in our switch architecture to achieve rate guarantees, and not just an optimization.

BAA solves the assignment problem by dynamically allocating rates to queues in the system based on observed traffic patterns. It collects information on traffic arrivals and queue backlogs and periodically updates each ingress line card with the rates it allocates for every queue in the system. We refer to this fixed duration between rate updates as an *epoch*.

The duration of an epoch clearly has implications for both the algorithm performance and hardware implementation. The implementation benefits from a longer duration, as this reduces the communications bandwidth necessary between chips and the amount of logic necessary, since given enough time it is possible to perform the major steps of the algorithm in serial fashion. Our results show that the algorithm performs well over

[2] We don't define term "demand" precisely at this stage, leaving that to Section V-A. For now, think of "demand" as the arrival rate.

a large range of epoch durations, with an epoch duration on the order of a few thousand time-slots giving results sufficient to meet our performance goals. The hardware implementation has therefore been optimized for cost, not speed.

Nonetheless, within the range of feasible values, a shorter duration does moderately improve the algorithm performance, and so the duration of an epoch in the implementation scales with the total number of queues, groups, and ports used. We consider this a worthwhile optimization as it does not incur any additional hardware cost.

We start by describing an ideal BAA scheme that solves the problem exactly. Then, we discuss a practical implementation that approximates this ideal solution.

A. Demand Estimation

BAA requires each ingress line card to estimate and maintain arrival rate information for every queue, and periodically send this information to the BAA module. Each ingress line card estimates, for each of its queues, a long-term arrival rate using a simple exponentially weighted moving average (EWMA) filter, updating this estimate every epoch (T_e). If A_{ij} is the arrival estimate for the queue on ingress i belonging to queue group j, for which f_{ij} bits arrived in the previous epoch, then

$$A_{ij} \leftarrow \alpha f_{ij}/T_e + (1-\alpha)A_{ij}, \quad (1)$$

where α is the EWMA gain factor. The matrix of A_{ij} values estimated using Equation 1 is *ingress-substochastic*, which means that $\forall i \sum_j A_{ij} \leq L_i$, ingress i's line rate. We find that a fixed value of α between 0.25 and 1 works well.

However, an EWMA estimator suffers from the problem that a sudden traffic spurt takes several epochs to track, affecting the responsiveness of the system and the resulting latency. The problem is severe when a queue group with a large guaranteed rate has had a low arrival rate that suddenly increases, because the group has sufficient (sudden) demand that may take several epochs to track. Queue backlog also accumulates when the aggregate arrival rate to a queue group exceeds its SLA, and some or all of the excess does not get serviced due to egress contention. If not under a steady-state condition, decreasing the EWMA gain factor creates a demand based on arrivals with better memory of these previous arrivals, which could not be used in the short term but should be used in the long term. Increasing the gain factor, on the other hand, allows the system to respond more quickly to a rapid increase in arrivals.

To combat this problem, we use a second component in the demand estimate: the queue backlog remaining at the end of the previous epoch. If the queue length at the end of an epoch is q_{ij} and the egress line rate for group j is R, then the total demand estimate is calculated as

$$D_{ij} = A_{ij} + B_{ij}, \quad (2)$$

where $B_{ij} = q_{ij}/R$ is the rate required to clear the backlog. Thus, the total demand has an estimated part (A_{ij}) and a deterministic part (B_{ij}). The matrix of D_{ij} values is not ingress-substochastic in general if the B_{ij}'s are non-zero.

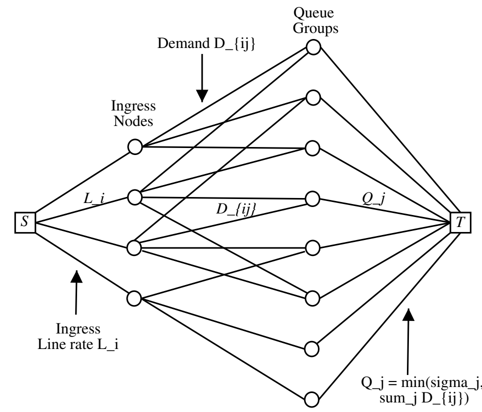

Fig. 3. Bipartite max-flow formulation of the bandwidth allocation algorithm for arbitrary demand values.

This complicates the process of assigning doubly substochastic g's to the different queue groups. The result of using both components in the demand estimate is that the system becomes more responsive to rapid changes in arrivals,

B. Satisfying Rate Guarantees

Given a matrix of demands D_{ij}'s, our goal is to assign g_{ij}'s to the queues such that all SLA guarantees are met. This problem is equivalent to a max-flow problem on a bipartite graph (Figure 3). One partition of the vertices corresponds to the ingress line cards and the other partition corresponds to the queue groups. The capacity of the edges connecting ingress i to queue group j is D_{ij}. We add a source vertex S and a terminal vertex T to the graph. The capacity of the edge from S to ingress i is L_i, the line rate of that ingress. The capacity of the edge connecting queue group j to T is $Q_j = \min(\sigma_j, \sum_i D_{ij})$. Any feasible flow on this graph is ingress-substochastic by construction.

Lemma 5.1: If the guaranteed rates σ_j's on any egress link do not oversell the link's bandwidth, and if the aggregate sum of egress link rates does not exceed the aggregate sum of ingress link rates, then the max-flow solution on the bipartite graph in Figure 3 saturates all the j-T links if and only if all queue group SLAs are met.

Proof: The key to the proof is to observe that the capacity of each j-T edge is picked as the smallest value that will satisfy the SLA. By definition, if every j-T link is satisfied, then all SLAs are met. For the "if" part of the claim, suppose the max-flow assignment of g_{ij} values does not satisfy some SLA. This means that the max-flow is smaller than the sum of all the j-T capacities. By construction, the sum of these capacities is the smallest cut in the graph, so this assignment of g_{ij}'s cannot be the max-flow since it violates the max-flow-min-cut theorem. ∎

Figure 4 shows the pseudocode of the BAA meeting rate guarantees.

The two vertex partitions in the bipartite graph have very different numbers of vertices (*e.g.*, 64 ingresses and 30,000

```
PROCEDURE BAA_SLA(A, D, Q, L)
// Matrix A is from Equation 1, D from Equation 2
// Q_j = min(σ_j, Σ_i D_ij); L_i = line rate of ingress i
∀ ingresses i { L'_i ← L_i; }
∀ queue groups j { want_j ← Q_j; }
∀ ingresses i, queue groups j {
    g_ij ← min(A_ij, (A_ij / Σ_i A_ij) Q_j);
    want_j ← want_j − g_ij;
    L'_i ← L'_i − g_ij;
}
∀ queue groups j
    if (want_j > 0) // SLA rate not yet met
        ∀ ingresses i { D'_ij ← D_ij − g_ij; };
∀ ingresses i : L'_i > 0, groups j : want_j > 0 {
    flow_ij ← MAXFLOW({i, L'_i}, {j, want_j}, {D'_ij});
    g_ij ← g_ij + flow_ij;
}
return G = {g_ij} matrix;
```

Fig. 4. Meeting SLA guarantees in BAA.

queue groups), so unlike a standard $O(nm)$ bipartite max-flow we can use a push-relabel method whose complexity is $O(n_s^2 m)$, where n_s is the number of vertices in the smaller of the two vertex partitions and m is the number of edges in the graph.

Depending on the switch size and the epoch duration, computing the max-flow exactly may not be possible. To make this more efficient, we observe that if the matrix of D_{ij} values is ingress-substochastic, then we can start by pushing D_{ij} units of flow through the edges without overfeeding any of the ingresses, dispensing with a full-blown max-flow computation. The result is a feasible solution to the max-flow. It is easy to find an ingress substochastic demand matrix—simply use the A_{ij}'s calculated in Equation 1 without using the B_{ij} estimates from Equation 2.

After this is done, many of the queue groups are likely to be saturated, so we temporarily remove these vertices from further consideration. For the remaining vertices, we create a subgraph of the original graph of Figure 3 where the assigned feasible capacities are subtracted from the links they traverse. This graph is likely to be substantially smaller than the original graph, and an approximate or exact max-flow algorithm can run on this.

Section V-D outlines a practical hardware implementation that approximates the ideal BAA described above.

C. Sharing Excess Bandwidth

BAA allocates the unallocated (excess) bandwidth on each egress link to the K queue groups using that link in weighted max-min-fair fashion according to an *excess allocation vector* $\phi = [\phi_1, \phi_2, \ldots, \phi_K]$. The input to this step includes the remaining demand (after BAA_SLA() runs) on each queue group, $D'_{ij} = D_{ij} - g_{ij}$ and the excess bandwidth on each egress link.

If the D'_{ij} values are such that $\sum_j (D'_{ij} + g_{ij}) \leq L_i$ for each ingress i, then this is a relatively straightforward problem because we don't have to worry about maintaining ingress-substochasticity of the final allocations. In this case, if $D'_j = \sum_i D'_{ij}$ is the aggregate demand of queue group j for the excess bandwidth on egress link k, and if there are K queue groups sharing that link, then the excess bandwidth on that link R'_k is easy to allocate in weighted max-min-fair fashion to the queue groups. It is possible to implement this in $O(K \log K)$ time per egress link.

In general, if we include the backlog estimates B_{ij} in D_{ij}, the D'_{ij}'s can't simply be shared as explained previously because ingress-substochasticity will not hold in the final solution. Our solution to this is to modify the D'_{ij} estimates to ensure that ingress substochasticity holds (this takes time linear in the number of queue groups) and then run the excess step.

Finally, BAA updates the allocations of rates to the individual queues in a queue group in proportion to their D'_{ij} values.

D. An Implementation

We have developed a hardware implementation of BAA that departs from the ideal maxflow-based solution in that it first forms an ingress-substochastic demand matrix from both the arrivals and backlog. The SLA guarantees can then be met without the max-flow step, since the backlog component is already included in the demand. Starting with an ingress-substochastic demand also means that excess allocation can proceed directly from the demand remaining after meeting SLA guarantees. (The ideal solution must make the remaining demand ingress substochastic at this point.)

This approach dramatically reduces the logic and epoch duration, at the expense of accuracy. In general, making the demand ingress substochastic in an accurate manner is a non-trivial problem, and requires information from other nodes in the system. Instead, to get the backlog component of demand for each queue, we simply take the remaining line rate after summing the arrivals for all of the queues, and allocate it to the queues in proportion to their lengths:

$$D_j = A_j + \frac{length(q_j)}{\sum_j length(q_j)} * (R - \sum_j A_j) \qquad (3)$$

The main drawback of this approach is that the process of making the demand ingress-substochastic may take demand away from some *egresses*, keeping some egress links underallocated. Reducing the demand in this manner does not take into account the contribution from other ingresses, or the bandwidth of the egress link to which the queue data is destined. That means that sometimes the backlog component will raise the demand above what a queue group can allocate, essentially wasting demand at that ingress that another queue may have been able to use.

Another difficulty with this approach is that it is possible for a queue that is consistently serviced below its arrival rate to claim an inordinate share of the backlog. To prevent this problem, the maximum demand that a queue may request above its arrival rate in any epoch is limited to a fixed amount above the rate allocated to that queue in the previous epoch.

These drawbacks may occasionally result in a queue group getting less than its SLA-specified guarantees even when there is adequate demand, or may lead to bandwidth sharing amongst queue groups different from the excess allocation vector,

Despite these shortcomings, this approach does not lead to under-utilization. This is because the HSA crossbar arbitration is work-conserving, and an egress link is kept idle only if there are no packets in the system for that egress.[3]

VI. Simulation Results

We present several simulation results in this section. We use a switch simulator that faithfully emulates the actual switch system down to the granularity of a single time-slot. This emulation includes the behavior of the line cards, the crossbar and its arbiter, the communication between the line cards and the arbiter, and the communication with the bandwidth allocator. Our simulator has been used for both hardware development and performance evaluations, and the results are a good indicator of system performance.

We have run simulations for a large number of different configurations and report on the results of one set with $N = 24$ line cards, speedup 2, link speed of 10 Gbps, TSbits = 1000 bytes, 100ms of ingress line card buffering, and an egress FIFO size of 100,000 bytes. Our simulator includes a traffic generator that generates synthetic traffic based on different distributions. We use it primarily to evaluate how well our architecture can handle overloads and how accurate the achieved bandwidths are relative to guarantees. We also evaluate BAA using Internet backbone traffic traces collected at a public peering point.

A. HSA Performance

The theoretical justification of HSA's correctness is an asymptotic rate result, and it is important to understand the time-scales over which HSA works well. We are interested in how accurate the achieved rates are over different measurement time-scales, and in how good the overload protection offered by HSA's tiered maximal matching is.

To answer these questions, we first need to define a metric for accuracy. Over any period of time W, if the bandwidth guarantee for a queue group is σ, then the number of bits sent should be at least σW if the queue group had sufficient demand. If over this period of time the group managed to send S_W bits, then we define

$$\text{error}(W) = \max(1 - \frac{S_W}{\sigma W}, 0). \quad (4)$$

We show the results for an $N \times N$ switch ($N = 24$) configured with 24 different queue groups per egress link.

[3] The product does allow network operators to set upper rate limits, which would of course prevent any enqueued packets from being sent if the rate limit were already met.

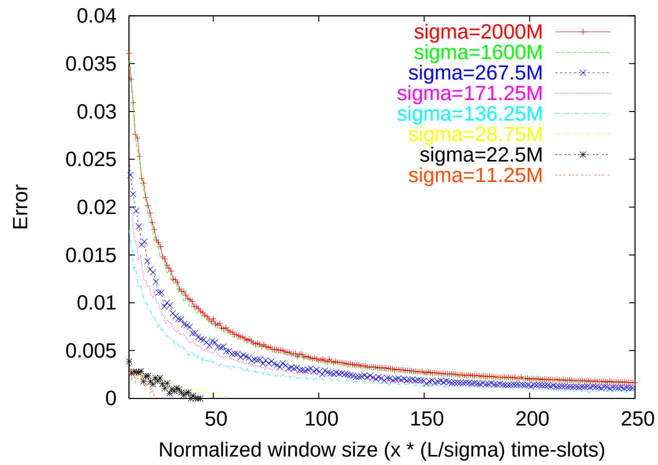

Fig. 5. Error in the non-overfed case as a function of time-window W for 9 queues with bandwidth guarantees between 2 Gbps and 11 Mbps.

Each queue group is a single queue, and no two queue groups sharing an egress are on the same ingress. Queue group i, comprising the queue on ingress i, has a guaranteed rate of $2 \times (4/5)^i$ Gbps, for $0 \leq i \leq 23$. These queue groups share an egress link of $L = 10$ Gbps. The sum of the guarantees of these 24 queues is 9.94 Gbps; we introduce another queue on one of the ingresses with guaranteed rate 60 Mbps, so all 10 Gbps are guaranteed. The incoming traffic on each group is a constant bit rate (CBR) with a packet interarrival duration determined by the queue's offered load. This simulation also tests HSA's ability to work across three orders of magnitude of rate guarantees—the highest rate is 2 Gbps, the lowest is 11 Mbps.

1) Non-overfed case: In the first experiment, the system is not overfed—the incoming traffic on each queue is a CBR that is equal to the guaranteed rate and does not overfeed any egress link. As expected, the guarantees are met. The errors for 9 of the queues are shown in Figure 5, which plots the error of different queues, calculated as the mean of different non-overlapping windows over an entire trace lasting about 250,000 time-slots. The x-axis in these plots is the window size normalized by L/σ rather than raw time-slots. For any queue that has a guarantee of σ, the average number of time-slots over which one time-slot's worth of packets leaves the system is L/σ. Hence the natural time-scale over which to evaluate error is to normalize the measurement window in terms of L/σ. The figure shows that for this wide range of guarantees, a window size of about 40 is enough to bring the error down to smaller than 1%. As expected, larger windows cause the error to go down towards 0.

The low-rate queues have less than 0.5% error even for small windows. For the larger-rate queues, the error is about 4% at very small windows, but this happens because some small windows may not have enough arrivals (and the error is computed assuming that at least σW bits are available). The interesting experimental result is that we have always been

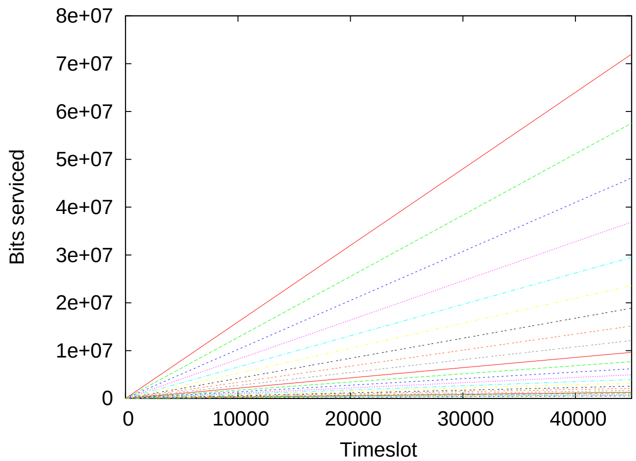

Fig. 6. Time sequence of serviced bits in the overfed case for the 24 different queue groups. The $L = 10$ Gbps egress link has an aggregate offered load of 216 Gbps, but is still able to isolate queue groups according to the configured guarantees.

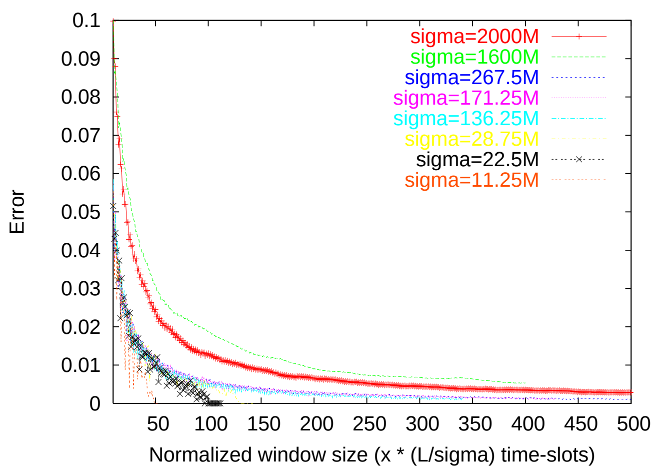

Fig. 7. Error in the overfed case as a function of time-window W for 9 queues with bandwidth guarantees between 2 Gbps and 11 Mbps.

able to obtain an error of less than 1% when the window size is about $50 \cdot L/\sigma$ time-slots, over a wide range of σ values in many different configurations.

2) Overfed case: Our goal is to "slam" the switch with significantly higher traffic than it can handle without substantial queueing or dropping and see if the configured guarantees are met. We use the same setup as in the previous case, except that each of the 24 queues has traffic arriving at 9 Gbps, all intended for a 10 Gbps link. This is substantial overload—216 Gbps arriving for a 10Gbps link. This extreme setup is intended to investigate and demonstrate that HSA's simple tiered-maximal matching with CCU is robust.

Figure 6 shows the sequence of packets that leave the egress link for the 24 different queues. The isolation between the queues and the difference in rates is clear. We observed similar results when the queues in the different groups share ingress links.

Figure 7 shows the error as a function of the normalized time window. The accuracy of HSA's guarantee depends on the degree of overload, but even in this extreme simulation most of the queues are within 1% of the rate guarantee within about 100 normalized time windows. As in the underfed case, the errors converge faster to 0 for lower-rate queues compared to higher-rate ones, suggesting that the errors when the number of queues is large may in fact be smaller.

B. BAA Performance

The performance of BAA depends on its ability to accurately estimate demand in the face of variation in traffic arrivals. To evaluate this, we analyze real-world traces. These traces are from a public peering link on the west coast of the United States. We analyzed data collected at different times in December 2002 and January 2003 from an OC-12 link (622 Mbps). We show the results of demand estimation over 36,000 time-slots (about 6 minutes) of one of the traces here. The results for longer periods of time and across different traces from this link are similar.

We conduct a pessimistic evaluation of BAA's ability to track variations in traffic by assuming that the traffic stream belongs to a queue group whose guaranteed bandwidth is infinite, so any arrivals should immediately be serviced. Since BAA estimates demand and ensures that the allocated rate g does not exceed the estimate, an estimator that underestimates the arrival rate will cause unnecessary delays. We note that this evaluation reflects both the ideal BAA and the hardware implementation, because the demand estimation is identical in both methods.

We logged the backlog remaining at the end of every 10ms epoch for two arrival estimators: pure EWMA with $\alpha = 0.25$ (Equation 1) and the EWMA-backlog sum (Equation 2). The performance of the EWMA+backlog scheme was substantially better; the average queue backlog remaining at the end of each epoch because not enough service was given to the queue was 3,431 bytes in this case, compared to 49,921 bytes for pure EWMA. On a different trace collected on a fast Abilene link, the average backlog for EWMA was 85,734 bytes compared to 5,729 bytes for the combined estimator.

Figure 8 shows the backlog between epochs 21,500 and 22,500 for the first trace discussed above. It shows a sudden spike in the arrival rate, and the resulting backlog at the end of that epoch. Including the backlog in the demand estimate quickly eliminates the backlog.

VII. CONCLUSION

This paper described the design of an input-queued switch system and its associated arbitration and rate allocation algorithms that achieve both absolute rate guarantees and proportional bandwidth sharing even under overloaded or adversarial traffic. Our algorithms are simple and scalable and require a switch speedup of 2 to provide rate guarantees. Our simulation results showed the robustness of the HSA to overload and a

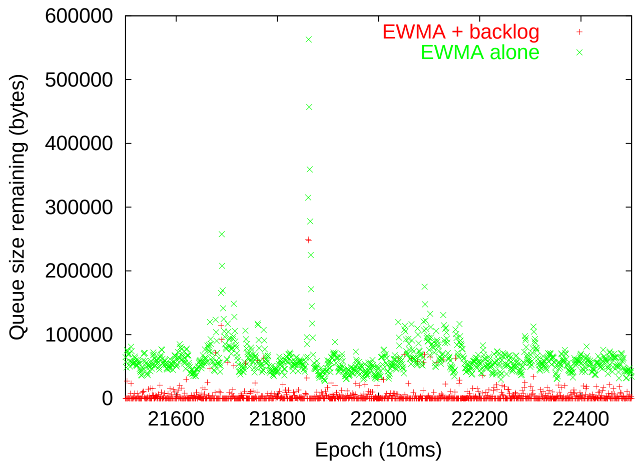

Fig. 8. Including the backlog handles sudden arrival rate spikes.

trace analysis showed the effectiveness of BAA in tracking arrival changes.

A semiconductor chipset that implements variants of these algorithms for routers with an aggregate capacity of 160 Gbps with links up to 10 Gbps is now commercially available, and a second-generation chipset supporting 640 Gbps will be available soon.

VIII. Acknowledgments

The authors benefited from technical discussions with several members of the Sandburst's Hibeam team, including Krste Asanovic, Stephen Bailey, Ken Dennen, Nick Horgan, Leo Keegan, Jongpil Lee, Tony Martuscelli, Rishiyur Nikhil (whose help with the simulations we also gratefully acknowledge), Daniel Rosenband, Karen Schramm, Jacob Schwartz, Eric Spada, Joe Stoy, and Scott Winterble. We thank Vince Graziani and Gary Kraemer for their encouragement and support of this work. We also thank the anonymous reviewers for their useful comments on this paper.

References

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulations of a Fair-Queueing Algorithm," *Internetworking: Research and Experience*, vol. V, no. 17, pp. 3–26, 1990.

[2] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proc. ACM SIGCOMM*, Cambridge, MA, 1995, pp. 231–242. [Online]. Available: citeseer.nj.nec.com/shreedhar95efficient.html

[3] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching Output Queueing with a Combined Input Output Queued Switch," *IEEE Journal on Selected Areas in Communication*, vol. 17, pp. 1030–1039, 1999.

[4] S. Iyer, R. Zhang, and N. McKeown, "Routers with a Single Stage of Buffering," in *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002, pp. 251–264.

[5] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output queued switching," Stanford University, Tech. Rep. CSL-TR-97-738, 1997. [Online]. Available: citeseer.nj.nec.com/prabhakar97speedup.html

[6] A. Charny, "Providing QoS Guarantees in Input Buffered Crossbar Switches with Speedup," Ph.D. dissertation, Massachusetts Institute of Technology, 1998.

[7] P. Krishna, N. Patel, A. Charny, and R. Simcoe, "On the Speedup Required for Work-conserving Crossbar Switches," *IEEE Journal on Selected Areas in Communication*, vol. 17, pp. 1057–1066, 1999, Presented at the 6^{th} IEEE/IFIP IWQOS'98, May 1998.

[8] I. Stoica and H. Zhang, "Exact Emulation of an Output Queueing Switch by a Combined Input Output Queueing Switch," in *Proc. IWQoS*, Napa, CA, May 1998, pp. 218–224.

[9] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *Proc. INFOCOM (2)*, 2000, pp. 556–564. [Online]. Available: citeseer.nj.nec.com/dai00throughput.html

[10] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[11] N. McKeown, V. Ananthram, and J. Walrand, "Achieving 100% throughput in an input queued switch," in *Proc. INFOCOM (2)*, 1996.

[12] E. Leonardi, M. Mellia, M. Marsan, and F. Neri, "Stability of Maximal Size Matching Scheduling in Input-Queued Cell Switches," in *Proc. International Conference on Communications (ICC)*, vol. 3, New Orleans, LA, 2000, pp. 1758–1763.

[13] N. McKeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[14] A. Kam and K.-Y. Siu, "Linear Complexity Algorithms for QoS Support in Input-Queued Switches with no Speedup," *IEEE JSAC*, vol. 17, no. 6, pp. 1040–56, June 1999.

[15] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Transactions on Computer Systems*, Nov. 1993. [Online]. Available: citeseer.nj.nec.com/anderson93high.html

[16] D. Stiliadis and A. Verma, "Providing Bandwidth Guarantees in an Input-Buffered Crossbar Switch," in *Proc. INFOCOM*, Boston, MA, April 1995.

[17] D. Stephens and H. Zhang, "Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture," in *Proc. INFOCOM*, 1998. [Online]. Available: http://www-2.cs.cmu.edu/~hzhang/papers/INFOCOM98b.pdf

[18] J. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*. Boston, MA: Kluwer Academic Publishers, 1990.

[19] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, "On Service Guarantees for Input Buffered Crossbar Switches," in *IWQoS'99*, 1999, pp. 79–86.

[20] ——, "Birkhoff-von Neumann Input Buffered Crossbar Switches," in *Proc. INFOCOM (3)*, 2000, pp. 1614–1623. [Online]. Available: citeseer.nj.nec.com/article/chang00birkhoffvon.html

[21] C. Koksal, R. Gallager, and C. Rohrs, "Rate Quantization and Service Quality over Single Crossbar Switches," in *Proceedings of IEEE INFOCOM*, Hong Kong, 2004.

[22] F. M. Chiussi, A. Francini, G. Galante, and E. Leonardi, "A Novel Highly-Scalable Matching Policy for Input-Queued Switches with Multiclass Traffic," in *Proc. IEEE GLOBECOM Conference*, 2002, pp. 2281–2286.