AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenanted Data Centers

Abdul Kabbani*, Mohammad Alizadeh*, Masato Yasuda †, Rong Pan‡, and Balaji Prabhakar*
*Department of Electrical Engineering, Stanford University
†System IP Core Research Laboratories, NEC Corporation, Japan
‡Cisco Systems, San Jose, California
{akabbani, alizade, balaji}@stanford.edu, m-yasuda@ct.jp.nec.com, ropan@cisco.com

***Abstract*—Data Center Networks represent the convergence of computing and networking, of data and storage networks, and of packet transport mechanisms in Layers 2 and 3. Congestion control algorithms are a key component of data transport in this type of network. Recently, a Layer 2 congestion management algorithm, called QCN (Quantized Congestion Notification), has been adopted for the IEEE 802.1 Data Center Bridging standard: IEEE 802.1Qau. The QCN algorithm has been designed to be stable, responsive, and simple to implement. However, it does not provide weighted fairness, where the weights can be set by the operator on a per-flow or per-class basis. Such a feature can be very useful in multi-tenanted Cloud Computing and Data Center environments.**

This paper addresses this issue. Specifically, we develop an algorithm, called AF-QCN (for Approximately Fair QCN), which ensures a faster convergence to fairness than QCN, maintains this fairness at fine-grained time scales, and provides programmable weighted fair bandwidth shares to flows/flow-classes. It combines the QCN algorithm developed by some of the authors of this paper, and the AFD algorithm previously developed by Pan et. al. AF-QCN requires no modifications to a QCN source (Reaction Point) and introduces a very light-weight addition to a QCN-capable switch (Congestion Point). The results obtained through simulations and an FPGA implementation on a 1Gbps platform show that AF-QCN retains the good congestion management performance of QCN while achieving rapid and programmable (approximate) weighted fairness.

I. Introduction

Multi-tenanted data centers host a large number of distinct tenants on a shared infrastructure. These data centers need to share computing, storage, and networking resources among the tenants in a fair and *programmable* manner. In recent years, advances in virtualization technologies have made great strides towards solving the problems of sharing computing and storage resources [8], [9]. However, progress in virtualizing the network has been limited.

Network virtualization entails providing programmable bandwidth guaranties to the different tenants, and ensuring performance isolation among the tenants. The performance of one tenant's traffic must not suffer due to the traffic of another; specifically, an aggressive, perhaps malicious, tenant must not be allowed to deny other tenants their fair share of the bandwidth resources. Also, ideally, isolation should not be achieved at the cost of poor network utilization, as can occur with static bandwidth reservations. It is, therefore, desirable to perform bandwidth allocation only at the onset of congestion when utilization is high. The notion of weighted max-min fairness satisfies all these requirements: different tenants are assigned weights or priorities, and each tenant's share of the network bandwidth is determined by it's weight or priority. Any unused bandwidth is recursively divided among the remaining tenants (again in weighted max-min fashion).

In this paper, we design and evaluate Approximately Fair QCN (AF-QCN). AF-QCN is a simple lightweight enhancement to the Layer 2 end-to-end congestion management algorithm, QCN, recently adopted for the IEEE 802.1 Data Center Bridging standard [12]. AF-QCN borrows ideas from the Approximate Fair Dropping algorithm of Pan et. al. [1], and extends the functionality of QCN, by adding an Approximate Fairness (AF) controller to the QCN switch.[1] As will be demonstrated via simulations (Section IV-B), and a hardware implementation (Section IV-C), this merger retains all the good properties of QCN, while enabling (approximate) weighted max-min fairness at the level of a few milliseconds (see below).

It is worth mentioning an interesting scheme, called E2CM [15], which was proposed at the IEEE 802.1Qau standards discussion. E2CM achieves fair bandwidth partitioning simultaneously with congestion management via path probing and destination-based per-flow fair share rate calculation. Because it combines fairness and congestion management, E2CM must be applied to all flows, regardless of whether they are large enough to cause congestion. Moreover, weighted bandwidth partitioning, if possible, seems difficult to achieve. By contrast, AF-QCN is a switch-side function that runs on top of QCN to provide weighted fairness to subset of flows or classes. Further, the flows that are monitored for providing bandwidth slices can be the few long-lived, elephant flows.

Congestion Management in Ethernet. The QCN (Quantized Congestion Notification) algorithm [13], [16], has been developed for the IEEE 802.1Qau standard [12], to provide end-to-end congestion management in switched Ethernet. In brief, QCN enables a switch to control the packet sending rate of an Ethernet source whose packets are traversing the switch.

In designing QCN, the major goals were:
(i) *Stability:* Queue occupancies should not oscillate widely, to avoid underflows causing link underutilization, and overflows

[1]This is similar to the XCP protocol, which decouples *efficiency control* (control of utilization or congestion) from *fairness control* [3]. In our case, efficiency control is achieved by the QCN control loop, and fairness control is provided by the AF algorithm (see Section III).

leading to packet drops.
(ii) *Responsiveness:* QCN should rapidly adapt source rates to extreme link bandwidth variations.
(iii) *Simplicity:* The algorithm should be very simple, as it is typically implemented in hardware.

Besides these primary goals, inter-flow fairness was also considered. Specifically, QCN flows must not suffer from systematic unfairness, which means that two flows starting with different sending rates, should have the same average rate[2]. However, the fairness of QCN is on a coarse time-scale (see Section IV-B and [17]), and, by design, it cannot provide programmable weighted fairness.

Approximate Fairness. One canonical method for providing fairness in a network has been to deploy schemes like Deficit Round Robin (DRR) or Weighted Fair Queueing to obtain packet-level fairness [4], [5]. However, such schemes require intricate packet scheduling algorithms, per-flow queues, and come at a high hardware implementation cost. In particular, requiring separate queues imposes hard limits on how many different classes can be supported, and is a major drawback of these approaches in multi-tenanted data centers. For example, major cloud providers today host tens of thousands of tenants [6], [7], while most switches only support 8 or 16 class of service queues.

In [1], Pan et. al. showed that by relaxing the packet-by-packet fairness requirement, and aiming for approximate fairness on longer time scales (on the order of several round-trip times), a much simpler algorithm called Approximate Fair Dropping (AFD) can be used. AFD is an active queue management scheme, which probabilistically drops packets based on queue size measurements (congestion information) *and* the estimated rate of individual flows (fairness information). The packet drops are used by responsive flow sources, such as TCP senders, as a congestion indication, prompting them to adjust their rates accordingly. The simplicity of AFD, and it's "soft per-flow state" requirement (only the rates of the flows needs to be monitored, as opposed to needing separate queues), allow it to be much more flexible than fine-grained packet scheduling algorithms, such as DRR. In fact, AFD can overlay on top of DRR to provide a behavior consistent with a DRR system that has a larger number of queues. The effectiveness of AFD in providing approximate fairness, and its low complexity design, make it amenable to high speed implementations, and have spurred its wide deployment in several switch and router platforms in industry [2].

Preliminary Description of AF-QCN. AF-QCN inherits several features from the AFD design with two major differences:
(i) In AF-QCN, congestion management is already well-handled by QCN, so the AF component is designed only to enforce fairness, leaving attaining stability to QCN.
(ii) The rate control provided by QCN is based on switch-to-source Congestion Notification Messages; switches send explicit messages to sources, asking them to cut their rates by a factor proportional to the value (denoted F_b) in the feedback message. In AF-QCN, we take advantage of this explicit feedback facility; rather than differentially dropping packets to obtain fairness as in AFD, AF-QCN adjusts the value of the feedback messages based on the fairness information provided by the AF component (see below and Fig. 1 for an example). Hence, on the one hand, the source rates are rapidly controlled by explicit feedback, and, on the other hand, intentional and undesirable packet drops at the switch are avoided.

[2]QCN has this property because the sampling of packets at a QCN switch (for congestion notification) is naturally biased to sampling high rate flows, and reducing their rates more frequently (see Section II-A).

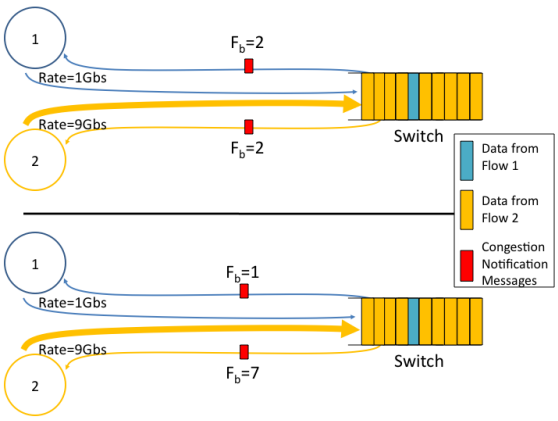

Fig. 1. QCN (above) vs AF-QCN (below) – Basic example: Unlike QCN, AF-QCN distinguishes between the 1Gbps and 9Gbps flows, and sends them different values in the Congestion Notification Messages.

Fig. 1 illustrates a basic example showing how AF-QCN differs from QCN. Two flows with different starting rates (1Gbps and 9Gbps) share the same bottleneck. Because QCN only takes the present and past queue size samples into consideration, it sends the same feedback value to the two flows. AF-QCN, however, distinguishes between the two flows based on their (estimated) sending rates, and adjusts the feedback to each flow accordingly. In this example, the amount of data that the switch receives (within a sampling interval) from flow 2, is 9 times that of flow 1. Therefore, AF-QCN sends a larger feedback value to flow 2 (who is exceeding its 50% fair share), and a smaller feedback value to flow 1 (who is below its fair share). Of course, the same mechanism can be used to provide programmable bandwidth allocation, as will be described in detail in Section III.

Rest of the paper. The paper is structured as follows: in Section II, we briefly review the QCN and AFD algorithms; in Section III, we describe the AF-QCN algorithm in detail; the simulation and hardware results are provided in Section IV; and we conclude in Section V.

II. Background

A. The QCN Algorithm

We now briefly describe the QCN algorithm, focusing on those aspects which are relevant to this paper (see [13] and [12] for more details). The algorithm is composed of two parts:
(i) *Switch or Congestion Point (CP) dynamics*: the mechanism by which a switch buffer attached to an oversubscribed link

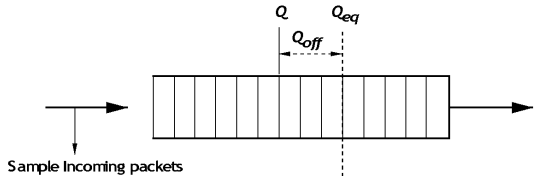

Fig. 2. Congestion detection in QCN CP buffer

samples incoming packets and generates a feedback message addressed to the source of the sampled packet. The feedback message contains information about the extent of congestion at the CP.

(ii) *Rate limiter or Reaction Point (RP) dynamics*: the mechanism by which a rate limiter (RL) associated with a source decreases its sending rate based on feedback received from the CP, and increases its rate *voluntarily* to recover lost bandwidth and probe for extra available bandwidth.

Congestion Point

The CP buffer is shown in Fig. 2. The goal of the CP is to maintain the buffer occupancy at a desired operating point, Q_{eq}. The CP samples incoming packets with an inter-sampling period depending on the severity of congestion[3]. With each sample, a congestion measure F_b is computed as follows: Let Q denote the instantaneous queue-size and Q_{old} denote the queue-size when the last packet was sampled. Let $Q_{off} = Q - Q_{eq}$ and $Q_\delta = Q - Q_{old}$. Then F_b is given by the formula:

$$F_b = Q_{off} + wQ_\delta, \quad (1)$$

where w is a positive constant (set to 2 by default).

The interpretation is that F_b captures a combination of queue-size excess (Q_{off}) and rate excess (Q_δ). Thus, when $F_b > 0$, either the buffer or the link or both are oversubscribed. A feedback message containing F_b, quantized to 6 bits, is sent to the source of the sampled packet *only* when $F_b > 0$; nothing is signaled when $F_b \leq 0$.

Reaction Point

The basic RP behavior is shown in Fig. 3. The RP algorithm has the following phases:

(i) *Rate decrease:* This occurs only when a feedback message is received. The RP decreases its sending rate (denoted CR for Current Rate) multiplicatively in proportion to the value of F_b received; the larger the F_b, the more CR is decreased.

(ii) *Fast Recovery (FR):* Immediately following a rate decrease episode, the RP enters the FR phase. In FR, the RP tries to reclaim the bandwidth lost in the decrease episode. It does this by successively increasing the Current Rate toward the *Target Rate (TR)*—the rate just before the decrease episode.

(iii) *Active Increase (AI):* After 5 increase cycles in FR have completed, the RP enters the AI phase where it probes for extra bandwidth on the path. In this phase, the RP increases its sending rate by adding a constant R_{AI} (5Mbps by default) to CR in each cycle.

[3]For example, the inter-sampling period is 150KB at low congestion, and can be as small as 18.5KB at high congestion.

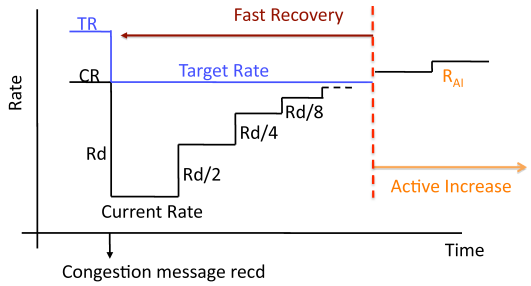

Fig. 3. QCN RP operation

As is evident in the above description, the RP performs rate increases in cycles. The durations of the cycles are determined based on the combination of a *Byte Counter*, which counts cycles in terms of the number of bytes transmitted by the RP, and a *Timer*, which counts cycles in terms of the amount of time elapsed.[4] The details of how the Byte Counter and Timer cycles determine the RP rate increase cycles can be found in [13].

Remark 1. QCN also has a *Hyper-Active Increase* phase to quickly seize unused bandwidth, when it suddenly becomes available [13].

B. The AFD Algorithm

AFD is an active queue management scheme that aims to control bandwidth allocation among flows/classes[5] that share a common queuing system. Controlled bandwidth allocation is achieved by probabilistically dropping the packets of classes for which the sending rate r_i is more than the fair share $rfair$.[6] As such, a key aspect of AFD is that unlike other active queue management systems (e.g. RED) where the decision to drop a packet is based solely on the queue depth, in AFD, it also depends on the sending rate of the packet's class.

Let $D_i = (1 - rfair/r_i)_+$ denote the probability with which a packet from class i should be dropped. Thus, if $r_i < rfair$ no drop will occur. If $r_i > rfair$, the drop probability increases as r_i gets further away from $rfair$. As a result, the throughput of each flow, $r_i(1 - D_i)$, is bounded by its fair share: $r_i(1 - D_i) = min(r_i, rfair)$. Hence, drops do not occur evenly across flows but are applied differentially to flows with different rates.

Two key algorithmic aspects of AFD lie in the manner in which r_i and $rfair$ are estimated via measurements. There are three elements in this procedure, as shown in Fig. 4.

Arrival Rate Estimation. Let M_i be the amount (measured in bytes, packets, etc) of traffic from flow i during the interval T_s.

[4]In the baseline implementation, each Byte Counter cycle is 150KBytes, and each Timer cycle is 15msec.

[5]In the rest of the paper, the terminology of flow and class will be used interchangeably; AFD (and AF-QCN) can be thought to provide fairness between flows or classes depending on the setting.

[6]For simplicity, in this section, we only consider the most basic version were all classes have an equal fair share. We will consider the more general case in the discussion of the AF-QCN scheme.

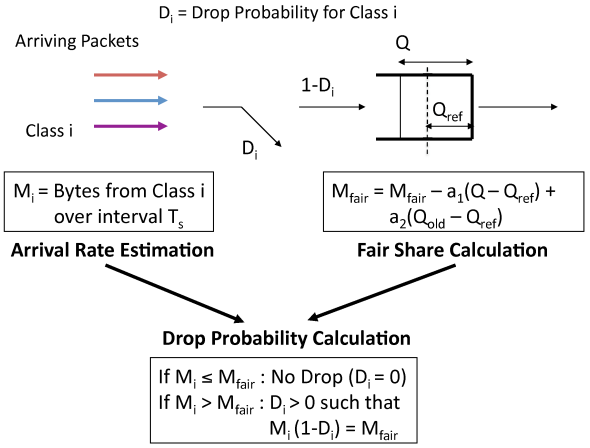

Fig. 4. Basic AFD Algorithm

Fair Share Calculation. Similarly, let $Mfair$ be the fair share amount of the traffic that the queue would have received from each flow, if the sources sent at their fair share rate. This is estimated dynamically, at the end of each measurement interval, as follows:

$$Mfair \leftarrow Mfair - a_1(Q - Q_{ref}) + a_2(Q_{old} - Q_{ref}), \quad (2)$$

where Q is the instantaneous queue length measured at the end of the current measurement interval; Q_{old} is the queue length measured at the end of the previous interval; Q_{ref} is the reference queue length (set by the operator); a_1 and a_2 are the averaging parameters (chosen as part of the design).

The main idea behind (2) is to try to dynamically find a value for $Mfair$ which along with the probabilistic dropping of packets by AFD, leads to the queue stabilizing around Q_{ref}. This allows AFD to perform bandwidth allocation and queue management simultaneously. For more details, see [1].

Drop Probability Calculation. The drop probability D_i for packets of flow i is chosen as:

$$D_i = (1 - Mfair_i/M_i)_+ \quad (3)$$

III. Approximate Fair QCN (AF-QCN)

A. Algorithm Description

The main goal of the AF-QCN algorithm is to enable an approximate bandwidth allocation on top of QCN. To this end, AF-QCN adds to the basic QCN Congestion Point, a fairness controller (henceforth called the AF controller), which is based on AFD. Recall that the basic QCN CP ensures that the link is well-utilized and the queue is stable around Q_{eq}. The AF controller ensures a weighted fairness among flows or flow classes without affecting the system stability.

More specifically, AF-QCN computes the feedback message value F_b as the weighted sum:

$$F_b = (1 - \alpha)F_{b-QCN} + \alpha F_{b-AF}, \quad (4)$$

where F_{b-QCN} is the original QCN congestion measure given by (1), and F_{b-AF} is a weighted fairness measure computed by the AF controller. As is the case with QCN, F_b is computed each time a packet is sampled, and is only sent if positive.

Remark 2. It is crucial to note that F_{b-QCN} is not a function of the flow: it only depends on congestion. However, F_{b-AF} does depend on the flow and is different for different flows.

Calculating F_{b-AF} involves the same three elements which exist in AFD and which we now detail.

Arrival Rate Estimation. This is nearly identical to AFD. We estimate the amount of traffic received during a time interval of T_s from each flow or flow class. The estimate for class i is denoted by M_i, and is updated every T_s seconds as follows:

$$M_i \leftarrow (1 - \beta)M_i + \beta M_{i-new}, \quad (5)$$

where $\beta \in (0, 1)$, and M_{i-new} is the actual amount of traffic from class i during the last T_s interval.

Fair Share Calculation. In AFD, $Mfair$ was dynamically estimated using (2). As mentioned in Section II-B, this was appropriate because as an active queue management scheme, AFD attempts to control the queue size around Q_{ref}, as well as perform bandwidth allocation. But AF-QCN is built on top of QCN, which already takes care of queue size management. Therefore, AF-QCN takes a more direct approach.

For each class i, let W_i denote its associated weight for bandwidth allocation. Then the fair share of class i is estimated as:

$$Mfair_i = \frac{W_i}{\sum_j W_j} \sum_j M_j. \quad (6)$$

$\mathbf{F_{b-AF}}$ Calculation. Given M_i and $Mfair_i$, the AFD drop probability would have been $D_i = (1 - Mfair_i/M_i)_+$. However, rather than dropping the packets of source i with probability D_i, AF-QCN incorporates D_i into the feedback message which will explicitly adjust the rate of source i. F_{b-AF} is computed by simply encoding the same D_i value as a 6 bit number:

$$F_{b-AF} = \lfloor 64 D_i \rfloor \quad (7)$$

Since $D_i \in [0, 1)$, note that $F_{b-AF} \in \{0, 1, \ldots, 63\}$. F_{b-QCN} is also a 6 bit number, and so F_b computed by (4) is a 6 bit number which will be fed back if positive.

Remark 3. A more robust estimation of the fair share can be obtained by only considering those flows/classes whose traffic exceeds a certain threshold. This ensures that short, transient flows do not skew the fair share for long flows. Specifically, (6) is replaced by:

$$Mfair_i = \frac{W_i}{\sum_{j \in A} W_j} \sum_{j \in A} M_j, \quad (8)$$

where $A = \{j : M_j > active\_thresh\}$ is the set of active flows. Of course, (8) is only used for flows $i \in A$. If a packet is sampled from a flow which is not active, F_{b-AF} is set to zero.

Remark 4. The situation is slightly more complicated when a maximum allowed rate, Max_i, can be prescribed for each class. In this case, the fair share values are set according to the max-min allocation, given the W_i's and Max_i's of the active flows.

B. Setting the Parameters

We now describe how the parameters α, T_s, β, and $active\_thresh$ are chosen.

The weight $\alpha \in (0,1)$ trades off stability for weighted fairness. Since stability is paramount, a small value of α ensures good stability by weighting F_{b-QCN} more than F_{b-AF}. An exact characterization of this trade off requires a control-theoretic study, which is left for future work. For now, based on extensive simulations, we choose $\alpha \leq 0.25$ for good stability.

The interval over which rates are estimated, T_s, must be large enough to allow robust rate estimates not affected by burstiness in packet arrivals (typically several round-trip times). However, it shouldn't be so large that convergence to fairness is overly delayed. Based on these considerations, assuming data center round-trip times are less than 500μsec, a T_s of 1 to a few milliseconds is recommended.

Finally, the parameters β and $active\_thresh$ should be chosen so as to allow small transient (mice) flows to pass without affecting the estimation of arrival rates and fair share values for long (elephant) flows. Therefore, they should be chosen with regard to the sizes we expect for the mice.

Following these considerations, the AF-QCN parameters $\alpha = 0.125$, $T_s = 1$msec, $\beta = 0.125$, and $active\_thresh =$ 20KB are chosen for all experiments in this paper.

C. Complexity

We discuss the "unoptimized complexity"[7] of the main operations that AF-QCN needs *in addition* to those of QCN. The operations are: (i) incrementing one M_{i-new} counter per arrival, (ii) updating the M_i value for all flows, and (iii) computing the $Mfair_i$ value per sampling interval, every T_s seconds. Hence, two registers are required per flow where the counter in (i) is accessed online via a hash table or a similar data structure. Because the computations at steps (ii) and (iii) need to be done once every T_s seconds, these can be done periodically and offline. Note that AF-QCN is meant to operate at the level of priority classes or long-lived/elephant connections. This essentially means that the state information required to be stored is small and the hardware complexity required to update the counters is fairly low.

IV. Evaluation

A. Preliminaries

We envision AF-QCN operating with QCN-compliant NICs. Each NIC potentially has a set of connections which can transmit at the full line rate, and a set of connections which are rate-limited and send through a QCN Reaction Point. A new connection begins at line rate, and is associated with a Reaction Point once it receives a Congestion Notification Message. It is de-associated once the rate of the Reaction Point reaches its maximum.

[7]There are several obvious optimizations which can be performed to significantly reduce the complexity of these operations. Indeed, some of these have been performed in AFD and implemented on hardware platforms. Due to a lack of space, and the obvious nature of the optimizations, we do not describe them here.

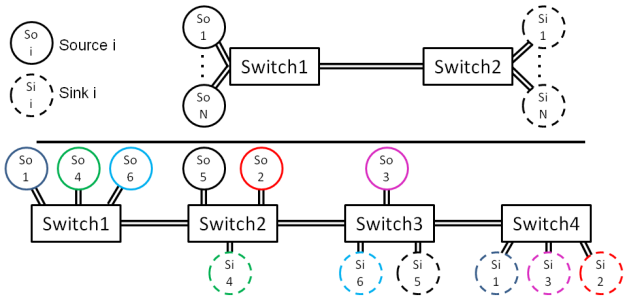

Fig. 5. Dumbbell (top) and parking lot (bottom) topologies

We simulate the AF-QCN design in various settings and under different network dynamics using ns2 [19]. We are interested in examining the performance of AF-QCN in the case where all the connections are backlogged (static flows), as well as in the case where we have a mix of static and dynamic flows. We compare the performance of AF-QCN with that of QCN using the two topologies in Fig. 5. All the links between the sources, sinks and switches are 10Gbps, and each switch hop involves a 50usec round-trip time delay (RTT), unless otherwise stated. The default QCN configuration [12] is used with 150KB switch buffers. Data packets are 1KB, and W_i and Max_i are set to 1 and 10Gbps respectively for all flows, except when indicated otherwise.

While our main interest is to study the bandwidth partitioning capabilities of AF-QCN, we also consider other important performance metrics such as queue size fluctuations and flow completion times.

B. Simulation Results

1) Static Flows:

Baseline experiment

We initiate 4 static flows traversing the single bottleneck in the dumbbell topology. The switch service rate is cut down from 10Gbps to 1Gbps at 2secs of simulation time and is increased to 10Gbps at 4secs. Fig. 6 shows the rates of the individual flows and the switch queue size plots (sampled every 10msecs) for QCN and AF-QCN. Notice how AF-QCN is successfully ensuring that all the flows are allocated their equal fair share rates at 10Gbps and 1Gbps, while maintaining the original queue size stability of QCN.

Long RTT

To ensure that stability and fairness are maintained as lags increase, we repeat the previous experiment with a 400usec RTT instead, which is considered to be very long for data center networks. The rates of the individual flows and the switch queue sizes plots are shown in Fig. 7 for QCN and AF-QCN.

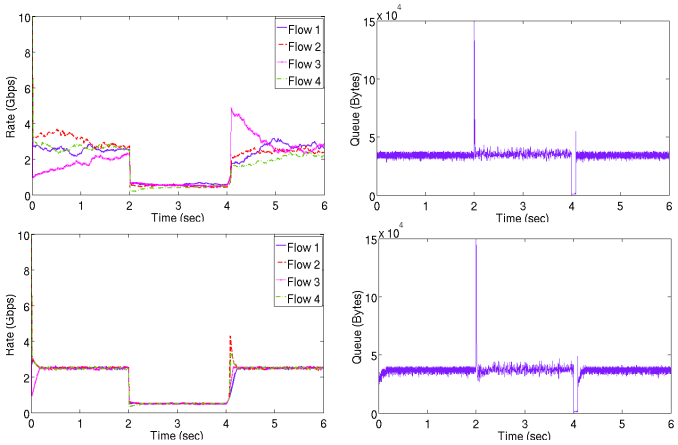

Fig. 6. Rates of the individual flows (left) and switch queue size (right) with QCN (top) and AF-QCN (bottom) at 50usec RTT

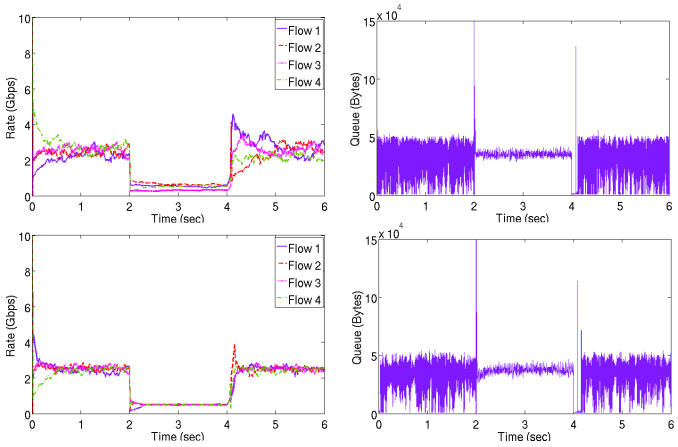

Fig. 7. Rates of the individual flows (left) and switch queue size (right) with QCN (top) and AF-QCN (bottom) at 400usec RTT

Increased multiplexing

To see the effect of increased multiplexing, we initiate 40 static flows simultaneously and keep them running for 6 seconds. The CDF of the individual flow rates normalized by 250Mbps (the fair share) is plotted in Fig. 8 for QCN and AF-QCN. The rates are measured every 10msec. Notice that with AF-QCN almost 99% of the flows fall within 25% of the fair share. With QCN, however, more than 45% of the flows are off by more than 25% of the fair share and around 10% are off by more than 50%.

Different weights and rate cap

We initiate 4 static flows as in the baseline experiment. However, this time the flows are given different weights: $W_1 = 4$, $W_2 = 3$, $W_3 = 2$, and $W_4 = 1$. We also set Max_1 to 1Gbps at 2sec to verify that AF-QCN can still maintain fairness among the three other flows at the ratio 3:2:1. Fig. 9 shows that the ratios of the rates of the individual flows are in accordance with the ratios of their weights before and after setting the 1Gbps cap on Flow 1.

Parking lot

Using the parking lot topology in Fig. 5, we incrementally

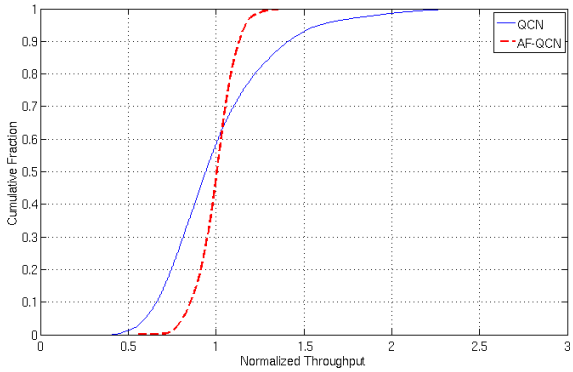

Fig. 8. CDF of flow rates with QCN and AF-QCN

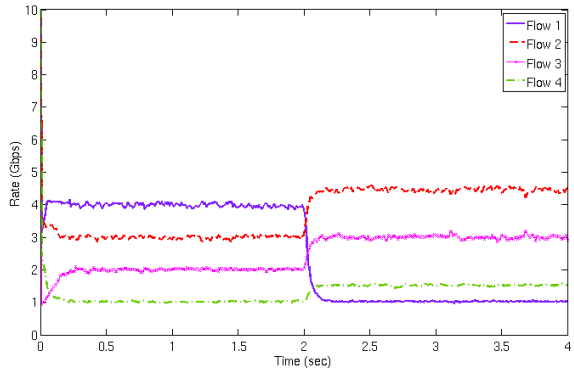

Fig. 9. Individual rates of 4 static flows with different weights

initiate each of the 6 static flows, one flow every second. Also, at 7sec, we set Max_1 to 1Gbps to cap the rate of Flow 1. Fig.10 shows the rates of the individual flows with QCN and AF-QCN.

Like TCP, QCN has a tendency to penalize multi-hop flows which have more congested links on their path. This is due to the relatively high frequency of Congestion Notification Messages such flows receive from multiple Congestion Points compared to those flows which pass fewer Congestion Points. In this topology, when the queues stabilize, most of congestion messages have small F_b values.[8] However, the cumulative effect of these messages adversely affect multi-hop flows and force their rates to be below their fair share.

AF-QCN ensures that this phenomenon does not occur. It compensates for the disparity in the frequency of Congestion Notification Messages for the many-hop and few-hop flows, by sending the few-hop flows larger F_b values when they exceed their fair share. Thus, AF-QCN guarantees the flows with fewer hops don't exceed their fair share.

In fact, not only does AF-QCN solve the unfairness caused to multi-hop flows, but the rate allocation it provides is the max-min fair allocation at each stage of the experiment. We have observed the same behavior in many other experiments not shown in this paper. An analysis of the nature of bandwidth

[8]In fact, most of the F_b values during this experiment where equal to 1.

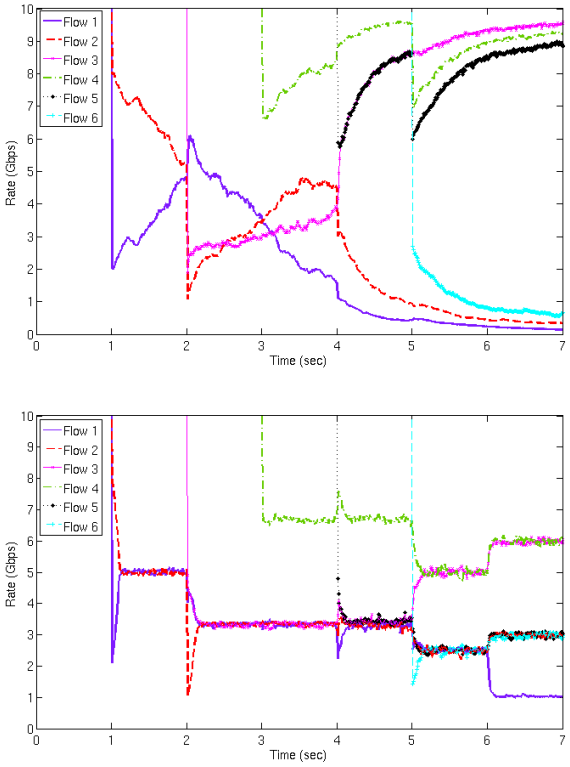

Fig. 10. Flows rates in the parking lot topology with QCN (top) and AF-QCN(bottom)

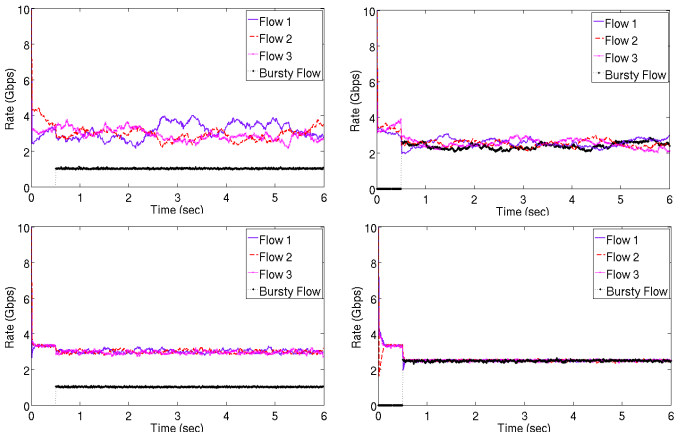

Fig. 11. Static and bursty flows throughput with QCN (top) and AF-QCN (bottom) at 1Gbps (left) and 6Gbps (right) offered load

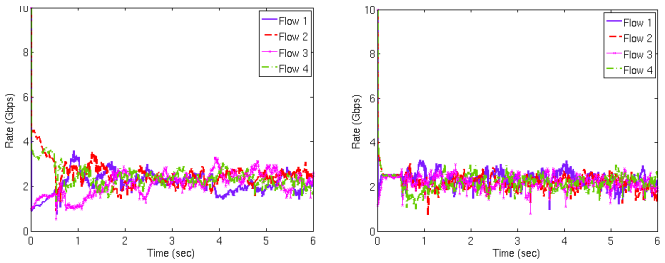

Fig. 12. Sending rates for long-lived flows in the presence of dynamic flow arrivals with QCN (left) and AF-QCN (right).

sharing achieved by AF-QCN in a general network topology is left for future work.

2) Mix of Static and Dynamic Flows:

Bursty connection

We initiate 3 static flows at time 0 and an on-off source (Bursty Flow) at time 0.5sec. The Bursty Flow is on for 10KB bursts, and its off duration is chosen so that its average offered load is 1Gbps in one experiment and 6Gbps in another experiment. We plot the rates of the individual flows in Fig. 11 for QCN and AF-QCN.

The bursty nature of the on-off source does not adversely affect the fairness among the 3 static flows. In the case where the Bursty Flow's offered load is 1Gbps, the other static flows are successfully able to utilize the remaining bandwidth. In the other case, where the Bursty Flow's offered load of 6Gbps is more than its fair share of 2.5Gbps, its throughput is limited to the 2.5Gbps fair share.

Flow completion time In this experiment, there are 8 QCN Reaction Points in total that share the single bottleneck in the dumbbell topology. The first 4 RPs each serve an infinitely back-logged static flow. The second set of 4 RPs each serve 4 *permanent* connections[9] (16 connections in total), over which (short) dynamic flows with Poisson arrival processes are initiated. The size of the dynamic flows is Pareto distributed with a mean of 10KB and a shape parameter of 1.1. The flow arrival rate is such that the total offered load from the dynamic flows is 1Gbps. We run the experiment until 100K flows are generated, and we investigate the effect of AF-QCN on the flow completion time of the dynamic flows.

As shown in Table I, the average FCTs for dynamic flows of different sizes is 30–50% smaller with AF-QCN when compared to QCN. This is because most of these flows are small, and are therefore not penalized by the AF controller. Hence, when they are sampled, the AF controller sends small F_b values, enabling them to finish faster. Note that this is correct behavior from the congestion point of view as well: small flows contribute very little to congestion. Furthermore, as Fig. 12 demonstrates even in the presence of dynamic flows, AF-QCN is able to maintain fairness between the long-lived flows.

TABLE I
FLOW COMPLETION TIME (FCT) WITH QCN AND AF-QCN

Flow Size Bin (KB)	**FCT with QCN (μs)**	**FCT with AF-QCN (μs)**
[1,10[	2.346	1.586
[10,100[	2.610	1.732
[100,1000[	5.037	2.932
[1000, ∞[	33.14	17.14

[9]These represent the permanent TCP connections common in data centers.

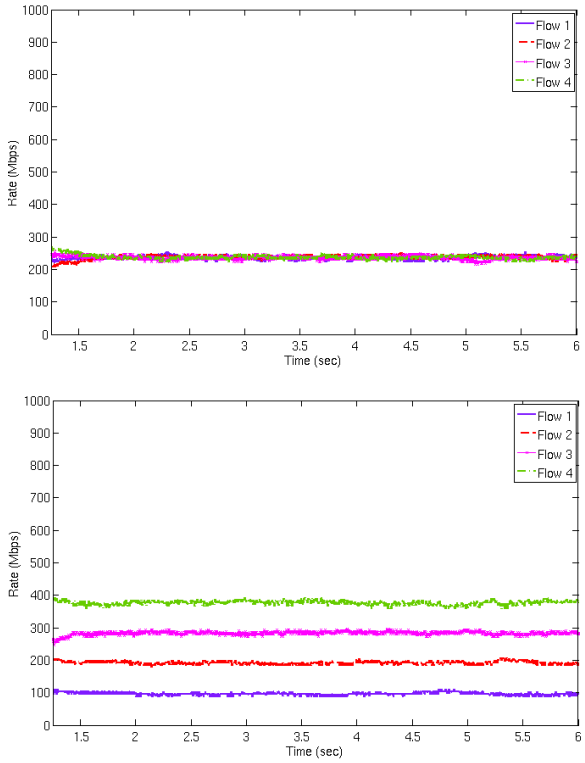

Fig. 13. AF hardware implementation - flow rates with equal weights (top) and different weights (bottom).

C. Hardware Implementation

We implement the AF design on top of our existing 1Gbps NetFPGA QCN switch [18] using the same AF-QCN parameters as mentioned previously and used in the simulations. The implementation is straightforward given the simplicity of the design. The correctness of this implementation is verified over the 50usec RTT dumb-bell topology in Fig. 5.

We run two experiments with 4 static flows sharing a QCN switch at a 950Mbps service rate. All the flows have the same weights in one experiment and weights in the ratio 1:2:3:4 in another experiment. The results are plotted in Fig. 13.

V. Conclusion

In this paper, we proposed and evaluated AF-QCN, an algorithm that adds a programmable bandwidth partitioning component based on AFD to the QCN Congestion Point mechanism. No changes are needed at a QCN Reaction Point. AF-QCN achieves weighted fairness at the granularity of a few milliseconds. This enables Data Center operators to provide programmable differential bandwidth allocation for flows or flow classes, a feature very useful in multi-tenanted Cloud Computing and Data Center environments. The results obtained via simulations and a hardware implementation show that AF-QCN retains the good properties of QCN (stability, responsiveness, and simplicity), while achieving rapid and programmable bandwidth partitioning.

One area for further work is a control theoretic study of AF-QCN for precisely characterizing the manner in which the AF component interacts with the QCN control loop, and the tradeoffs in setting the different parameters. Understanding the equilibrium fairness properties of AF-QCN in arbitrary networks also warrants further investigation.

Acknowledgment

Mohammad Alizadeh is supported by a Caroline and Fabian Pease Stanford Graduate Fellowship. The authors would like to thank the anonymous reviewers, whose comments helped improve the paper.

References

[1] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, “Approximate Fairness Through Differential Dropping”, Computer Communication Review, January 2003.
[2] R. Pan, B. Prabhakar, F. Bonomi, and B. Olsen, “Approximate fair bandwidth allocation: A method for simple and flexible traffic management,” *Allerton*, September 2008.
[3] Dina Katabi, “Decoupling Congestion Control and Bandwidth Allocation Policy With Application to High Bandwidth-Delay Product Networks,” Ph.D. Dissertation, Massechussetts Institute of Technology, March 2003.
[4] Shreedhar, M., and Varghese, G., “Efficient Fair Queueing using Deficit Round Robin”, ACM Computer Communication Review, vol. 25, no. 4, pp. 231242, October, 1995.
[5] Bennett, J. and Zhang, H., “Hierarchical Packet Fair Queueing Algorithms”, SIGCOMM Symposium on Communications Architectures and Protocols, pp. 143-156, August, 1996.
[6] http://aws.amazon.com/ec2/
[7] http://www.microsoft.com/azure/default.mspx
[8] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and Others. “Above the clouds: A Berkeley view of cloud computing.” Technical Report UCB/EECS-2009-28, Berkeley, 2009.
[9] R. Buyya, C. S. Yeo, and S. Venugopal. “Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities.” In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, 2008.
[10] http://www.t11.org/ftp/t11/pub/fc/bb-5/09-056v5.pdf
[11] http://www.ieee802.org/1/pages/dcbridges.html
[12] http://www.ieee802.org/1/pages/802.1au.html
[13] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman, “Data center transport mechanisms: Congestion control theory and IEEE standardization,” *Allerton*, September 2008.
[14] http://www.ieee802.org/1/files/public/docs2008/au-prabhakar-qcn-los-gatos-0108.pdf
[15] http://www.ieee802.org/1/files/public/docs2007/au-sim-IBM-ZRL-E2CM-proposal-r1.09b.ppt
[16] http://www.ieee802.org/1/files/public/docs2008/au-rong-qcn-serial-hai-pseudo-code%20rev2.0.pdf
[17] http://www.ieee802.org/1/files/public/docs2008/au-pan-qcn-benchmark-sims-0108.pdf
[18] http://www.ieee802.org/1/files/public/docs2009/au-kabbani-yasuda-0509-HW_implementation_evaluation.pdf
[19] Network Simulator. ns2. http://www.isi.edu/nsnam/ns.