AFCD: An Approximated-Fair and Controlled-Delay Queuing for High Speed Networks

CONFERENCE PAPER · JANUARY 2013

DOI: 10.1109/ICCCN.2013.6614103

6 AUTHORS, INCLUDING:

Lin Xue
Louisiana State University
9 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Cheng Cui
NetApp
8 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Chui-Hui Chiu
Louisiana State University
5 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Seung-jong Park
Louisiana State University
43 PUBLICATIONS **577** CITATIONS
SEE PROFILE

AFCD: An Approximated-Fair and Controlled-Delay Queuing for High Speed Networks

Lin Xue*, Suman Kumar†, Cheng Cui*, Praveenkumar Kondikoppa*, Chui-Hui Chiu*, Seung-Jong Park*
*School of Electrical Engineering and Computer Science, Center for Computation & Technology,
Louisiana State University, Baton Rouge, LA, USA
†Department of Computer Science, Troy University, Troy, AL, USA
Email: {xuelin, sumank, ccui, pkondi1, cchiu1, sjpark}@cct.lsu.edu

***Abstract*—High speed networks have characteristics of high bandwidth, long queuing delay, and high burstiness which make it difficult to address issues such as fairness, low queuing delay and high link utilization. Current high speed networks carry heterogeneous TCP flows which makes it even more challenging to address these issues. Since sender centric approaches do not meet these challenges, there have been several proposals to address them at router level via queue management (QM) schemes. These QM schemes have been fairly successful in addressing either fairness issues or large queuing delay but not both at the same time. We propose a new QM scheme called Approximated-Fair and Controlled-Delay (AFCD) queuing for high speed networks that aims to meet following design goals: approximated fairness, controlled low queuing delay, high link utilization and simple implementation. The design of AFCD utilizes a novel synergistic approach by forming an alliance between approximated fair queuing and controlled delay queuing. It uses very small amount of state information in sending rate estimation of flows and makes drop decision based on a target delay of individual flow. Through experimental evaluation in a 10Gbps high speed networking environment, we show AFCD meets our design goals by maintaining approximated fair share of bandwidth among flows and ensuring a controlled very low queuing delay with a comparable link utilization.**

I. Introduction and Motivation

To meet the demand of high speed networks, several TCP variants have been proposed. Extensive performance evaluations of these protocols convinced network users to adopt several of these seemingly promising proposals. Furthermore, open sourcing of these protocols make it flexible for user to select the choice of their protocols. Therefore, current computer network is ruled by heterogeneous TCP variants [1] such as TCP-SACK, HighSpeed TCP (HSTCP) [2], CUBIC TCP (CUBIC) [3], etc. TCP flows are dominant in Internet2 traffic [4] accounting for around 85% flows while rest being UDP flows with most of the network usages being long-lived bulk data transfer. One of the key design goals of the proposed TCP variants is fair bandwidth sharing with other competing TCP flows. Unfortunately, our study [5] discloses severe inter-protocol unfairness among heterogeneous long-lived TCP flows in high speed networks where faster TCP flows consume most of the network bandwidth, whereas slow ones starve.

Packet delay is an equally key high speed network performance measure with others being throughput and fairness. Although high speed networks do not have Bufferbloat problem [6], queuing delay in high speed networks needs careful consideration. For example, in a typical setting of a 10Gbps router [7] in high speed networks, the output buffer size of routers could be up to 89MB, which may create large queuing delay in high speed networks. The large queuing delay may create bad user experience in live concert [8], video streaming, etc, over high speed networks like Internet2. Also, popular applications such as Online shopping, Voice over IP, HDTV, banking, and gaming, require not only high throughput but also low delay. In fact, importance of packet delay is growing with the emergence of a new class of high performance computing applications such as algorithmic trading and various data center applications with soft real time deadlines that demand extremely low end-to-end latency (ranging from microsecond to orders of milliseconds). It is clear that the large latency may result in poor performance of the networks. Therefore, predictable and controllable queuing delay is highly desired in contemporary high speed networks.

Performance of heterogeneous TCP flows depends on router parameters [9], but high bandwidth, high latency, and bursty nature of high speed networks make it a challenging task to design queue management (QM) schemes that ensure minimal latency while maintaining fair share of bandwidth among heterogeneous flows. There have been a considerable effort to address these challenges through various QM proposals [10]–[13]. The QM scheme in [10] uses stateful information of the flows to classify the incoming packets, and puts the classified packets into different queues which is not suitable for large scale networks. The QM scheme in [11] is stateless in core layer, but it needs stateful information in edge layer. In [12], the authors proposed a stateless active queue management (AQM) solution, and it achieves approximate fairness for heterogeneous flows. Although the fairness problem has been addressed in these QM schemes, controllable and predictable queuing delay has been overlooked. Recently proposed controlled delay (CoDel) AQM scheme [13] focuses on providing extremely low queuing delay, and CoDel works well in a wide range of scenarios. While CoDel provides extremely low queuing delay, fairness issue is at bay. Only very recently that there is an attempt to bring fair queuing into CoDel and a variation of CoDel has been implemented in Linux kernel [14] that provides fairness by classifying different flows into dif-

978-1-4673-5775-3/13/$31.00 ©2013 IEEE

ferent CoDel queues. However, the history of research on classification based QM suggests that any approach requiring huge amount of state information is not practical for large scale networks.

With above key observations we are highly motivated to address fairness and queuing delay issues in high speed networks. In this paper, we propose an AQM scheme: Approximated-Fair and Controlled-Delay (AFCD) queuing that satisfies the requirement of high speed networks by providing approximated fairness and controllable low queuing delay. AFCD is a synergy of fair AQM design and controlled delay AQM design. The key idea of our novel AFCD queuing is to form an alliance between fairness based queuing and controlled delay based queuing. Proposed AQM uses a relatively small amount of state information to provide approximated fairness while ensuring very low queuing delay for high speed networks. AFCD achieves comparable throughput as other popular AQM schemes as well. Extensive evaluation of AFCD shows that AFCD performs well in a 10Gbps high speed networking testbed CRON [15].

The rest of the paper is organized as following. In section II, we present background and related works of this paper. Section III is the design of AFCD. We evaluate AFCD in Section IV, and we draw our conclusion in Section V

II. Background and Related Work

Drop-tail (DT) is the most used QM in commercial router nowadays. Packets are served in the order of first in first out. When DT queue is full, packets are simply dropped at the tail. However several studies have shown the limitations that imposed by DT. AQM scheme is suggested to eliminate those limitations. The authors in [16] compare AQM with DT and shows that AQM have a minor impact on the aggregate performance metrics and AQM is sensitive to traffic characteristics that may compromise their operational deployment.

Random early detection (RED) [17] is an AQM scheme. Packets are dropped early and randomly before the queue is full. RED has two thresholds: min threshold and max threshold. When the exponentially average queue size is smaller than the min threshold, no packet is dropped. When the exponentially average queue size is bigger than the max threshold, all packets are dropped. When the exponentially average queue size is between min threshold and max threshold, packets are marked/dropped based on a probability which is increased linearly with the queue size. RED has its own limitations. Level of congestion and the parameter settings affect the average queue size and the throughput is also sensitive to the traffic load and to RED parameters. Various modifications have been suggested to improvise RED. DRED with multiple packet drop precedence to allow differentiating traffic based on priority [18], Gentle RED [19] with smooth dropping functions and many more.

Besides, totally new AQM schemes are also proposed in different studies such as BLUE [20] which uses packet loss and link idle events, instead of queue length, to manage congestion. Stochastic fair blue (SFB) [21] is an AQM for enforcing fairness among a large number of flows. SFB uses a bloom-filter to identify the non-responsive flows. The bloom-filter hashes the incoming packets to a hash value which stands for the flows. SFB maintains a mark/drop probability pm for each of the flows and a $qlen$ which is the number of queued packets belonging to the flow. SFB has two thresholds: max is the maximum length of $qlen$, $target$ is the desired length of $qlen$. If a flow's $qlen$ is larger than max, packets are dropped. If a flow's $qlen$ is smaller than max, packets are marked/dropped randomly with probability pm, meanwhile pm is adjusted to keep $qlen$ between 0 and $target$. If pm of a flow reaches 1, the flow is identified as non-responsive flow, and therefore SFB enters rate-limit function to rate-limit the non-responsive flows.

In next few paragraphs, we describe the two notable approaches namely, AFQ [12] and CoDel [13] which form the basis and background for our approach.

Approximate fair dropping queue (AFQ) [12] provides approximate fair bandwidth allocation among flows. AFQ makes the drop decision based on the sending rate of the flow. To estimate the sending rate of the flow, AFQ uses a shadow buffer and a flow table. The shadow buffer is used to sample the incoming packets. The flow table contains the packet count of the flow. AFQ estimates the flow's rate r_i and the fair share rate r_{fair}. r_i is estimated by m_i which is the amount of traffic from flow i during an interval. r_{fair} is estimated dynamically by m_{fair}, which is determined by:

$$m_{fair} \leftarrow m_{fair} + \alpha(Q_{old} - Q_{target}) - \beta(Q - Q_{target}) \quad (1)$$

where Q is the instantaneous queue length in current interval, Q_{old} is the queue length in previous interval, Q_{target} is the target queue length, α and β are the averaging parameters. The drop probability of a packet from flow i is denoted as:

$$D_i = (1 - r_{fair}/r_i)_+ \quad (2)$$

If $r_i < r_{fair}$, no packet is dropped. If $r_i > r_{fair}$, D_i is increased and packets are dropped based on D_i.

Recently proposed CoDel queue [13] provides extremely low queuing delay and aims to solve the Bufferbloat problem in the Internet. Unlike other AQM's complicated setting of parameters, CoDel is parameterless, and works efficiently for a wide range of scenarios. CoDel calculates the packet-sojourn time at the dequeue function and keeps a single-state variable of how long the minimum packet queuing delay has been above the $target$ value. If the packet queuing delay is larger than $target$ for at least $interval$, a packet is dropped and CoDel's control law is set for the next drop time. The control law sets the next drop time in inverse proportion to the square root of the number of drops happened since CoDel has entered $dropping$ state. If the packet queuing delay is smaller than $target$, CoDel's controller stops dropping.

In summary, all of these QM schemes aim to concentrate on some specific aspects of the network performance. However, none of these QM schemes provides fairness, very low queuing delay, and a considerable throughput at the same time for high speed networks.

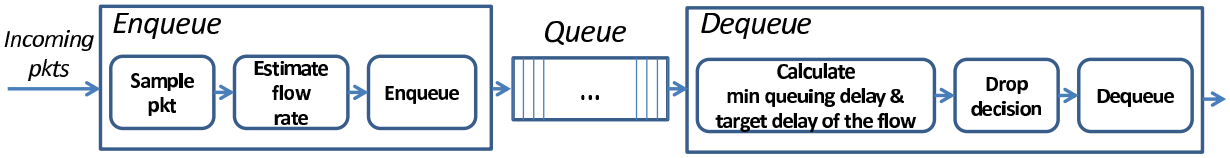


Fig. 1: Functional block diagram of AFCD queuing

III. The Approximated-Fair and Controlled-Delay Queuing

In this section, we present the idea and mechanism behind Approximated-Fair and Controlled-Delay queuing.

A. Design Goals

AFCD has the following key design goals:

Fairness: AFCD needs to provide approximately fair bandwidth allocation for the flows sharing a high speed bottleneck link. Here, the approximate fairness means the max-min fairness among long-lived flows.

Minimal Queuing Delay: In addition to providing approximate fairness, AFCD needs to provide a very low queuing delay for high speed networks. In terms of queuing delay, AFCD needs to perform as good as CoDel.

Acceptable Link Utilization: Like other AQM schemes, AFCD drops packets early and gently, which makes some sacrifices in terms of throughput. Here, by throughput we mean the long term throughput achieved by the flows in the bottleneck link. We do not claim that AFCD can provide the same throughput performance as DT, but AFCD needs to have similar throughput performance to other AQM schemes.

Simple Implementation: Our design of AFCD aims to facilitate fair share among flows while keeping queuing delay very low in high speed networks environment. According to the statistics of high speed networks [4], most of the network resources are consumed by long-lived bulk data transfer and the number of the long-lived flows is small. Thus, it is feasible for us to take an approach similar to AFQ's to estimate the flow's sending rate by using shadow buffer and flow table, which only require a very small amount of state information. When we make drop decision at dequeue function, the single-state variable of delay information is also extremely light-weighted to implement.

Architecture, design, and algorithm of AFCD follow next.

B. Architecture

Architecture of AFCD is presented in Fig. 1. When a packet comes to the enqueue function of AFCD, we sample the packet based on a sample interval. As presented in AFQ [12], packet sampling is good enough for the queue to obtain enough state information to estimate the rate for long-lived flows. A shadow buffer and a flow table is used for estimating the flow's sending rate information. The shadow buffer is used to keep the sampled packets. The flow table contains the flow's packet count. The shadow buffer and the flow table do not need to be big, because the number of long-lived flows in high speed networks is small [4]. Thus, AFCD only maintains a small amount of flow state information. If the packet is sampled, the packet is hashed based on its flow information tuples. We update the shadow buffer and the flow table by the hash value. The packet is mapped into the shadow buffer, and the flow table is also updated by increasing or decreasing the flow's packet count. Therefore, AFCD gets enough state information to estimate the sending rate of the long-lived flows.

After we get the flow's approximate rate information from the enqueue function, we make the drop decision at the dequeue function based on each packet's queuing delay. We only require single-state variables for a minimum delay within an interval and a target delay for individual flow. This is similar to the approach which CoDel takes. AFCD makes the drop decision at dequeue function to get the information of packets queuing time, and therefore AFCD obtains accurate packets queuing time information and controls queuing delay very well.

C. Design

With the flow table, we know flow i's bandwidth share m_i is proportional to flow i's packet count in flow table:

$$m_i \leftarrow flow\_table[i] \tag{3}$$

The fair bandwidth share m_f can be estimated by the shadow buffer size, flow count and an averaging value of m_f:

$$m_f \leftarrow \frac{SHADOW\_BUFFER\_SIZE}{flow\_count} \tag{4}$$

$$m_f(t2) \leftarrow m_f(t1) + \alpha(Q(t1) - Q_{target}) - \beta(Q(t2) - Q_{target}) \tag{5}$$

where Equation 5 is the same as in AFQ [12]. t1 is the previous time, t2 is the current time, Q is the instantaneous queue length, and α and β are the averaging parameters.

In AFCD's dequeue function, there is a $target\_delay$ that needs to be set by network operators the same as in CoDel. Within an $interval$, if the minimum queuing delay is higher than this $target\_delay$, packets are dropped. Based on this $target\_delay$, AFCD sets different target delay for different flows to approximately enforce fair bandwidth allocation among the flows. Flow i's target delay $target\_delay_i$ is calculated by flow i's bandwidth share and fair bandwidth share as following:

$$target\_delay_i \leftarrow target\_delay \times (\frac{m_i}{m_f})^{-3} \tag{6}$$

We ran experiments to find the value of $target\_delay_i$ in the above equation. Thus, if flow's bandwidth share is equal

to the fair bandwidth share, flow's target delay is the target delay. If flow's bandwidth share becomes lower than the fair bandwidth share, flow's target delay becomes much higher than the target delay, and therefore packets from the slow flows are not dropped. If flow's bandwidth share becomes higher than the fair bandwidth share, flow's target delay becomes much lower than the target delay, and therefore the fast flows are penalized.

A packet from flow i is dropped because the queuing delay has exceeded $target\_delay_i$ for at least $interval$. Then AFCD controller is the same as CoDel's [13]. The controller dequeues next packet from the queue, and sets the next drop time which is decreased in inverse proportion to the square root of the number of drops since the last dropping state. Until the queuing delay of a packet from flow i is lower than $target\_delay_i$, AFCD stops dropping.

D. Algorithm

A pseudo code of AFCD algorithm is shown below:

Algorithm of AFCD

```
shadow_buffer[SHADOW_BUFFER_SIZE]
flow_table[FLOW_TABLE_SIZE]

function AFCD_QDISC_ENQUEUE
    if sample_packet() then
        Calculate hash for packet, afcd_hash
        Use afcd_hash to update shadow_buffer
        Use afcd_hash to update flow_table
    end if
    do_enqueue()
end function

function AFCD_DEQUEUE
    Calculate hash for packet, afcd_hash
    Use afcd_hash and flow_table to calculate target delay of the
flow i, target_delay_i
    while within interval, minimum_queuing_delay >
target_delay_i do
        qdisc_drop()
        Dequeue next packet
        Schedule next drop time
    end while
    Stop drop
end function
```

In enqueue function, AFCD samples packet by using $sample\_packet()$. Then AFCD updates $shadow\_buffer$ and $flow\_table$ by the hash value of the packet $afcd\_hash$.

In dequeue function, AFCD calculates the target delay of flow i $target\_delay_i$ and the minimum queuing delay $minimum\_queuing\_delay$ within interval $interval$. Then AFCD makes its drop decision.

IV. Experimental Evaluation

In this section, we evaluate AFCD carefully, and compare AFCD to other related QM schemes such as DT, RED, SFB, CoDel, and AFQ. We conduct various experiments to emulate various scenarios in high speed networks.

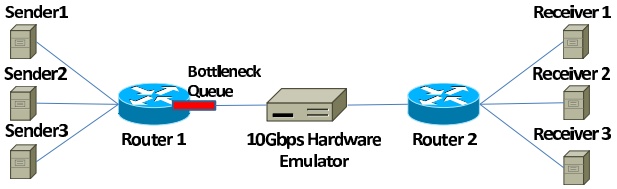


Fig. 2: Dumbbell experimental topology with 10Gbps environment

We setup a dumbbell networking topology in a 10Gbps high speed networking testbed CRON [15] as shown in Fig. 2. All servers and links in the topology have 10Gbps capacity. The bottleneck queue is the output queue of Router1, where we set the queue management schemes. Three pairs of Senders and Receivers run a modified Linux 2.6.34 kernel, and are used to transfer TCP/UDP flows by using Zero-copy Iperf [22]. Two routers run a modified Linux 3.6.6 kernel, which supports all related Linux queue disciplines including newly developed CoDel. The 10Gbps hardware emulator is used to accurately emulate propagation delay. We initially set the propagation delay in the hardware emulator to be 120ms. Queue size in the bottleneck queue is set to be 100% of bandwidth-delay product (BDP).

In CRON, we did system tunning for 10Gbps high speed networking environment as described in [5], [23]. In addition, we implement AFQ and AFCD queue disciplines in Linux kernel as well as user-space tc command. For AFCD, we have initial setting of parameters as CoDel [13]. $target\_delay$ is set to 5ms and $interval$ is set to 100ms. $target\_delay_i$ will be calculated by AFCD algorithm.

A. 1 CUBIC flow and 1 TCP-SACK flow

We first send 1 CUBIC flow from Sender1 to Receiver1 and 1 TCP-SACK flow from Sender2 to Receiver2. The experiment ran for 300 seconds. Fig. 3 shows the results of throughput and delay for all QM schemes. Fig. 3(a) shows DT, RED, and CoDel have serious unfairness when CUBIC and TCP-SACK are competing with each other in a 10Gbps bottleneck. SFB alleviates unfairness a little, but TCP-SACK still gets much less throughput than CUBIC. AFQ and AFCD show a good fairness performance, where CUBCI and TCP-SACK have almost the same throughput. Fig. 3(b) shows the maximum queuing delay created by all the QM schemes. DT creates extremely high queuing delay. RED, SFB, and AFQ all create a queuing delay larger than 15ms. CoDel and AFCD show an extremely low queuing delay, which is less than 1ms.

We see that AFCD performs the same as AFQ in terms of fairness, while AFCD performs the same as CoDel in terms of delay. To get a clear view of why AFCD has such kind of performance, we plot out the instantaneous congestion window (Cwnd) and instantaneous RTT of DT, CoDel, AFQ, and AFCD in Fig. 4. Fig. 4(a) and 4(e) show the performance of a DT QM scheme. DT makes unfairness between CUBIC and TCP-SACK, and a very high queuing delay. Fig. 4(b) and 4(f) show that CoDel makes unfairness between CUBIC and TCP-SACK, but CoDel keeps an extremely low queuing

delay. 4(c) and 4(g) are for AFQ. AFQ, on the other hand, treats CUBIC and TCP-SACK fairly, but AFQ makes some queuing delay.

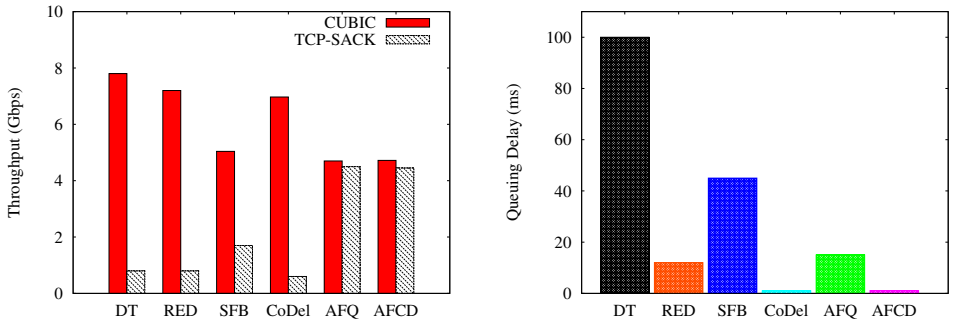
As shown in 4(d) and 4(h), AFCD makes approximated fair between two different TCP flows as well as low queuing delay. AFCD estimates the sending rate of the flows and sets different target delay for different flows based on the sending rate of the flows, and therefore both fairness and very low queuing delay are achieved. In case of AFQ and AFCD, there is a peak of bursty traffic at the beginning of the traffic, which is because that AFQ and AFCD need some time to gain the state information of the flows.

B. 1 CUBIC flow and 1 UDP flow

Next we start 1 CUBIC flow from Sender1 to Receiver1 and 1 UDP flow from Sender2 to Receiver2 simultaneously. The experiment also runs for 300 seconds. The UDP flow is a non-responsive flow sending at a speed of 10Gbps. Fig. 5 shows throughput and delay performance of the QM schemes. In Fig. 5(a), DT, RED and CoDel all have serious unfairness. CUBIC flow barely gets any bandwidth, but UDP flow consumes almost all of the bandwidth. SFB, AFQ, and AFCD make fairness between CUBIC and UDP. SFB has an improvement in fairness performance compared to Section IV-A because of its instinct to detect non-responsive flows. Fig. 5(b) shows that DT, RED, SFB, and AFQ all make different degrees of large queuing delay, but CoDel and AFCD keep the queuing delay less than 1ms. This test shows that AFCD works well in the presence of non-responsive flow.

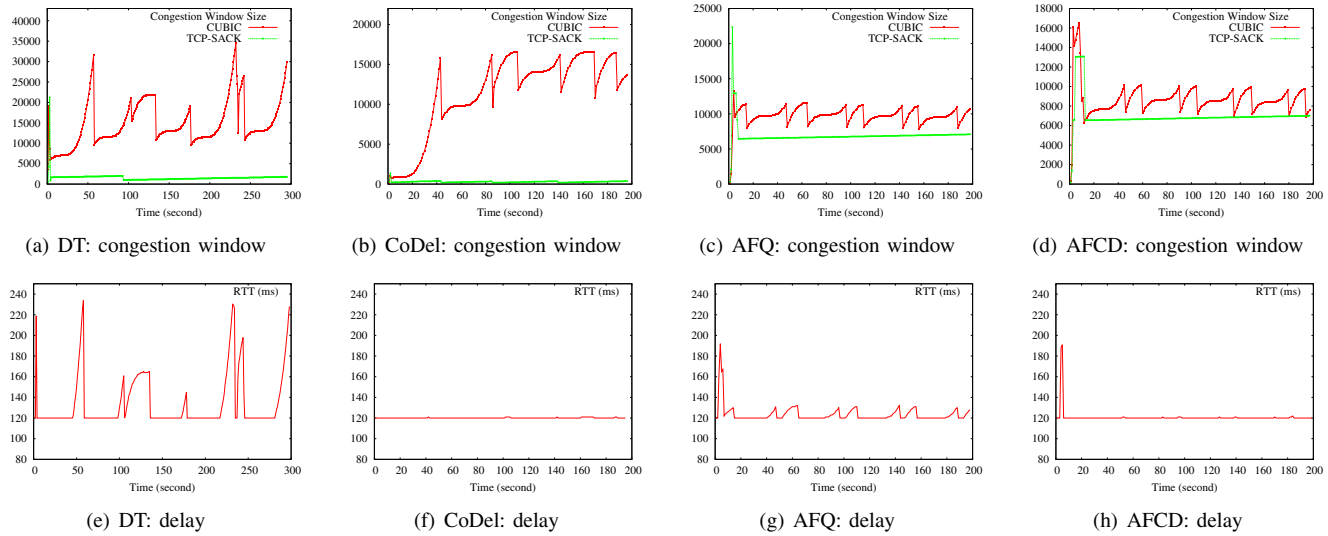
C. 3 flows case: 1 CUBIC, 1 HSTCP, and 1 TCP-SACK flow

In this section, we mix 3 of the most popular TCP variants [1] in the bottleneck. 1 CUBIC flow, 1 HSTCP flow, and 1 TCP-SACK flow are transfered between the 3 pairs of senders and receivers respectively. We run the experiments 5 to 7 times, and get the results in Fig. 6. Fig. 6(a) shows AFQ and AFCD get the highest Jains fairness index, while DT, RED, and CoDel perform bad in terms of fairness. In terms of queuing delay as shown in Fig. 6(b), CoDel and AFCD keep the delay very low, while DT gets the highest delay. Fig.



(a) Throughput of TCP flows (b) Maximum queuing delay

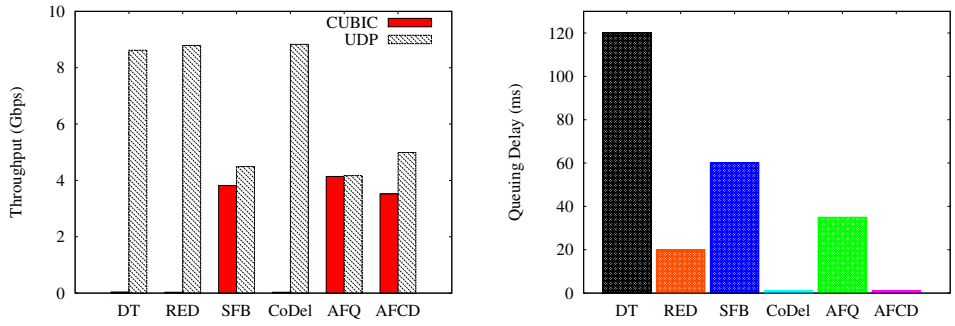
Fig. 3: 1 CUBIC and 1 TCP-SACK: performance of QM schemes when 1 CUBIC flow and 1 TCP-SACK flow are competing at the 10Gbps bottleneck, Propagation Delay = 120ms, Queue Size = 100% BDP



(a) DT: congestion window (b) CoDel: congestion window (c) AFQ: congestion window (d) AFCD: congestion window

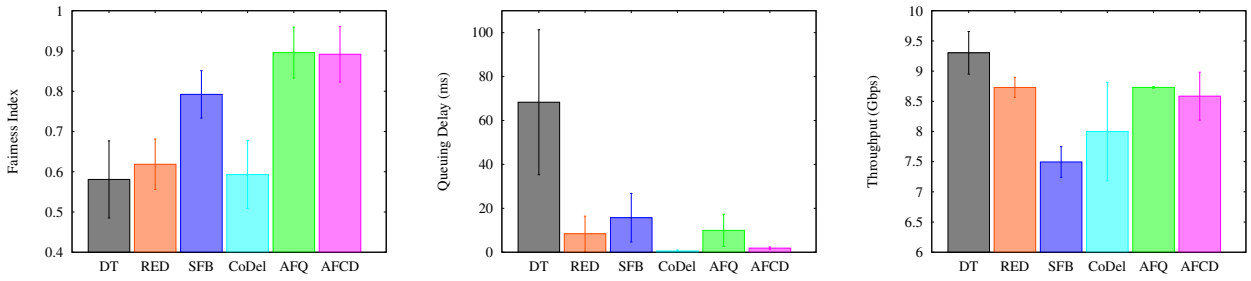
(e) DT: delay (f) CoDel: delay (g) AFQ: delay (h) AFCD: delay

Fig. 4: 1 CUBIC and 1 TCP-SACK: instant congestion window size and delay for different QM schemes
1 CUBIC flow and 1 TCP-SACK flow are competing at the 10Gbps bottleneck, Propagation Delay = 120ms, Queue Size = 100% BDP



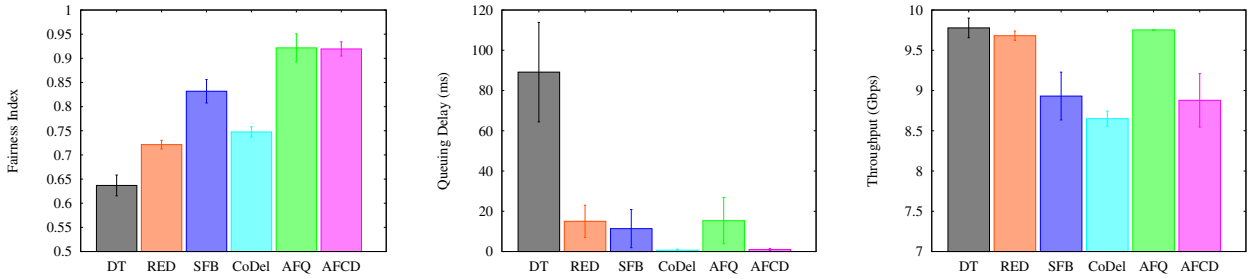
(a) Throughput of TCP/UDP flows (b) Maximum queuing delay

Fig. 5: 1 CUBIC and 1 UDP: performance of QM schemes when 1 CUBIC flow and 1 UDP flow are competing at the 10Gbps bottleneck, Propagation Delay = 120ms, Queue Size = 100% BDP



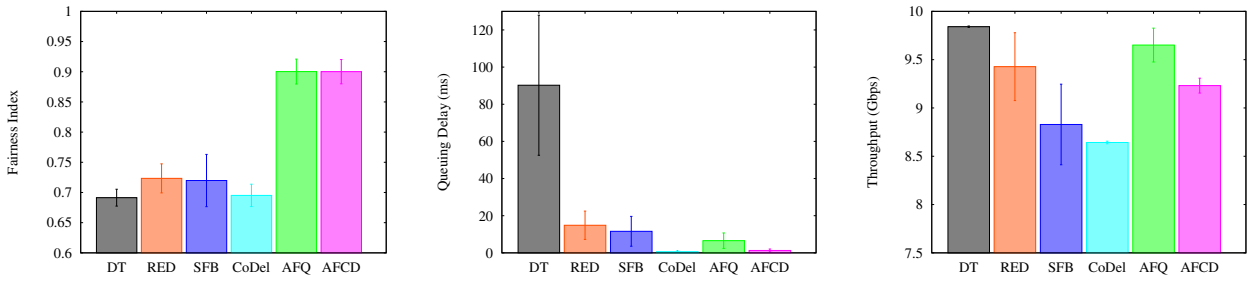
(a) Jain's fairness index (b) Average Queuing Delay (c) Throughput

Fig. 6: 3 flows case: performance of queue management schemes in 10Gbps high speed networks
1 CUBIC, 1 HSTCP, and 1 TCP-SACK flow, Propagation Delay = 120ms, Queue Size = 100% BDP



(a) Jain's fairness index (b) Average Queuing Delay (c) Throughput

Fig. 7: Many multiplexed heterogeneous TCP flows: performance of queue management schemes in 10Gbps high speed networks
10 CUBIC, 10 HSTCP, and 10 TCP-SACK flow, Propagation Delay = 120ms, Queue Size = 100% BDP



(a) Jain's fairness index (b) Average Queuing Delay (c) Throughput

Fig. 8: With short-lived TCP flows: performance of queue management schemes in 10Gbps high speed networks
10 CUBIC, 10 HSTCP, 10 TCP-SACK flow, and short-lived TCP flows, Propagation Delay = 120ms, Queue Size = 100% BDP

6(c) is the result of throughput. AFCD performs almost the same as RED and AFQ.

D. Many Multiplexed Heterogeneous TCP Flows

We increase the multiplexing of long-lived TCP flows in this test. 10 CUBIC, 10 HSTCP, 10 TCP-SACK flows are mixed in the 10Gbps bottleneck link. Fig. 7 shows the performance of the QM schemes in case of increased multiplexing. Fig. 7(a) shows fairness performance is improved in all QM schemes because of the multiplexing. AFQ and AFCD still get the highest fairness among all QM schemes. Fig. 7(b) is the delay results. CoDel and AFCD still create the least queuing delay. Fig. 7(c) shows AFCD has similar throughput performance to SFB and CoDel.

E. With short-lived TCP flows

In this experiment, we test the performance of QM schemes with both long-lived and short-lived TCP flows. We add a pair of sender and receiver, and use Harpoon traffic generator [24] to create two-way short-lived TCP flows in the bottleneck link. The inter-connection times from Harpoon TCP client to Harpoon TCP server follow exponential distribution with 1 second of mean. The request file sizes follow Pareto distribution with alpha=1.2 and shape=1500. Fig. 8(a) shows AFQ and AFCD still perform better in terms of fairness. Fig. 8(b) shows CoDel and AFCD still maintain very low queuing delay. Fig. 8(c) shows AFCD has similar throughput performance to other QM schemes in the presence of short-lived TCP flows.

V. Conclusion and Future Work

In this paper, we proposed a novel QM called Approximated-Fair and Controlled-Delay (AFCD) that aims to provide approximate fairness and very low queuing delay in a controlled and predictable manner for high speed networks. AFCD uses very small amount of state information of flows to approximately estimate the flows' sending rate when packets are enqueued. When packets are dequeued, AFCD uses a single-state variable to calculate a target delay of the flow and makes drop decisions for different flows based on the target delay of the flow. We evaluate the performance of AFCD in a 10Gbps high speed networking environment. Overall performance of AFCD is superior among its peers in various scenarios. In terms of fairness, AFCD performs as good as AFQ. Heterogeneous flows get approximated fair bandwidth share over AFCD. In terms of queuing delay, AFCD controls queuing delay to be very low, which is as good as CoDel. Also, AFCD gets comparable throughput performance to other AQM schemes. In the future, we plan to optimize the algorithm of AFCD to get better performance in high speed networks. We also plan to implement the AFCD algorithm in 10Gbps NetFPGA board.

Acknowledgment

This work has been supported in part by the NSF MRI Grant #0821741 (CRON project), GENI grant, and DEPSCoR project N0014-08-1-0856.

References

[1] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, "Tcp congestion avoidance algorithm identification," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 310–321.

[2] S. Floyd, S. Ratnasamy, and S. Shenker, "Highspeed tcp for large congestion windows," IETF, RFC 3649, 2003.

[3] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[4] "Internet2 Netflow Data," http://netflow.internet2.edu.

[5] L. Xue, S. Kumar, C. Cui, and S.-J. Park, "An evaluation of fairness among heterogeneous TCP variants over 10gbps high-speed networks," in *37th Annual IEEE Conference on Local Computer Networks (LCN 2012)*, 2012, pp. 348–351.

[6] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the internet," *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.

[7] Cisco, "Buffers, Queues, and Thresholds on the Catalyst 6500 Ethernet Modules," 2007, http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps708/prod_white_paper09186a0080131086.pdf.

[8] Internet2, "Global Concert Series over Internet2," 2009, https://k20.internet2.edu/projects/100.

[9] A. Tang, X. Wei, S. Low, and M. Chiang, "Equilibrium of heterogeneous congestion control: Optimality and stability," *Networking, IEEE/ACM Transactions on*, vol. 18, no. 3, pp. 844–857, 2010.

[10] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *Networking, IEEE/ACM Transactions on*, vol. 4, no. 3, pp. 375–385, 1996.

[11] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 33–46, 2003.

[12] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate fairness through differential dropping," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 23–39, 2003.

[13] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[14] E. Duzamet, "CoDel fair queuing," 2012, https://dev.openwrt.org/browser/trunk/target/linux/generic/patches-3.3/042-fq_codel-Fair-Queue-Codel-AQM.patch.

[15] CRON, "CRON Project: Cyberinfrastructure for Reconfigurable Optical Networking Environment," 2011, http://www.cron.loni.org/.

[16] G. Iannaccone, M. May, and C. Diot, "Aggregate traffic performance with active queue management and drop from tail," in *ACM SIGCOMM*, 2001.

[17] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 4, pp. 397–413, 1993.

[18] Aweya, M. Ouellette, A. Dasylva, and D. Y. Montuno, "Dred-mp: queue management with multiple levels of drop precedence," *International Journal of Network Management*, vol. 14, no. 6, 2004.

[19] V. Rosolen, O. Bonaventure, and G. Leduc, "A red discard strategy for atm networks and its performance evaluation with tcp/ip traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 3, pp. 23–43, Jul. 1999.

[20] W. chang Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The blue active queue management algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 4, 2002.

[21] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Stochastic fair blue: A queue management algorithm for enforcing fairness," in *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1520–1529.

[22] T. Yoshino, Y. Sugawara, K. Inagami, J. Tamatsukuri, M. Inaba, and K. Hiraki, "Performance optimization of TCP/IP over 10 gigabit ethernet by precise instrumentation," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 11.

[23] L. Xue, C. Cui, S. Kumar, and S. Park, "Experimental evaluation of the effect of queue management schemes on the performance of high speed tcps in 10gbps network environment," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*. IEEE, 2012, pp. 315–319.

[24] J. Sommers, H. Kim, and P. Barford, "Harpoon: a flow-level traffic generator for router and network tests," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1. ACM, 2004, pp. 392–392.