LRURC: A Low Complexity and Approximate Fair Active Queue Management Algorithm for Choking Non-Adaptive Flows

Xianliang Jiang, Guang Jin, and Jiangang Yang

***Abstract*—Active Queue Management is an effective way to improve the TCP's performance. However, due to the existence of non-adaptive flows, most previous schemes cannot achieve efficiency, fairness, and easy-to-deploy simultaneously. In this paper, a novel scheme, named LRURC, is proposed to improve the fairness among different flows and reduce the flow completion time when the high-bandwidth non-adaptive flows arise. Motivated by the CHOKe algorithm, the LRURC adopts the *sample-match* mechanism to identify high-bandwidth flows. Furthermore, the least-recently-used mechanism is introduced to manage the high-bandwidth flows. Different from most other schemes, the LRURC calculates the packet dropping probability according to the flows' rate, which is estimated by the *sample-match* mechanism. This can achieve fairness effectively and reduce the influence of high-bandwidth non-adaptive flows. When the flow is not hit by the *sample-match* mechanism, the packet dropping probability remains constant. Simulation results show that LRURC can improve the fairness among different flows, reduce the completion time of adaptive flows, and hold the queue length around an expected value when high-bandwidth non-adaptive flows arise.**

***Index Terms*—Active queue management, approximate fairness, virtual queue, least recently used algorithm.**

I. Introduction

ACTIVE Queue Management (AQM) for Internet routers plays an important role in congestion control. Its goal is to improve the TCP's performance, and balance the link utilization and delay. However, as the growth of various applications, the performance of adaptive flows (e.g., TCP) suffers a degradation significantly in the presence of sudden, non-adaptive cross-traffic (e.g., selfish UDP flows), which further affect the fairness of various flows. Therefore, the fair AQM schemes are proposed to alleviate the problem. In this paper, we focus on the fairness issue in the presence of high-bandwidth non-adaptive flows and propose an approximate fair AQM scheme to reduce the completion time of adaptive flows, and hold the queue length around the expected value.

In [1], Floyd *et al.* proposed a seminal AQM algorithm, the Random Early Detection (RED), which calculates the Packet Dropping Probability (PDP) according to the average queue length to acquire a better queue stability. Actually, RED is hard to ensure the fairness of different flows and protect the adaptive flows when the high-bandwidth non-adaptive flows arise. Thus several approximate fair AQM schemes [2], [3] were proposed to penalize aggressive non-adaptive flows and ensure the fair share of bottleneck bandwidth. Specifically, Pan *et al.* [3] proposed the stateless AQM scheme (named CHOKe), which uses the *sample-match* mechanism to generate the congestion signal for realizing approximate fairness in core routers. In [4], [5], the throughput, spatial characteristics, and transient behavior of CHOKe were analyzed and investigated. Moreover, Eshete *et al.* [6] introduced an extra flow matching trial (*max-comp*) to enhance the power of flows protection of CHOKe. However, due to the burstiness of network traffic and the agnostic PDP of CHOKe and its variants, some useful packets of adaptive flows are dropped, which affects the fairness and Flow Completion Time (FCT). To overcome the *bufferbloat* problem in bottleneck queues, Nichols *et al.* [7] developed an AQM algorithm (named CoDel) by limiting the sojourn time of packets passing through routers. The FQ-CoDel in [8] combines packet scheduler and CoDel for reducing latency across the Internet. Recently, Zhang *et al.* [9] presented a Robust RED (RRED) against the Low-rate Denial-of-Service (LDoS) attacks. Jiang *et al.* [10] proposed a trusted RED (named RED-FT) to mitigate the impact of Flooding DoS (FDoS) and LDoS attacks. In brief, how to ensure the fairness of different flows, reduce the FCT, and protect adaptive flows is critical when the high-bandwidth non-adaptive flows arise.

In this paper, we present a low complexity and approximate fair AQM (named LRURC) using the least-recently-used (LRU) and the rate choke (RC) (*sample-match*) mechanism. It can maximize the approximate fairness among different flows, reduce the completion time of adaptive flows, and hold the queue length around the expected value. Different from the per-flow fair queueing (e.g., Enhanced Weighted Fair Queuing [11]), LRURC don't isolate flows into separate subqueues (logical or physical) and just needs to maintain *partial flow state* ($O(1)$ spatial complexity). Compared with CHOKe [3] and its variants (e.g., gCHOKe [6]), LRURC can better protect adaptive flows and enhance the fairness of different flows. Furthermore, for dealing with the unfairness, more complex variants of RED were also proposed to apply differential per-flow drop rates. But studies have shown that these variants were insufficient to protect adaptive flows effectively. Due to space limitations, we do not list all AQM schemes for analysis and refers to [12] for details.

II. The Design and Implementation of LRURC

We begin by presenting the key idea and queue management of LRURC, and then describe its key components. We also discuss the fairness of LRURC when the adaptive and non-adaptive flows coexist.

Manuscript received September 2, 2014; revised January 22, 2015; accepted January 23, 2015. Date of publication February 9, 2015; date of current version April 8, 2015. This work was supported in part by the Natural Science Foundation of Zhejiang Province (LY12F02013) of China; Ningbo Municipal Technology Innovation Team (2011B81002) of China. The associate editor coordinating the review of this paper and approving it for publication was G. Reali.

X. Jiang and J. Yang are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310023, China (e-mail: jiangxianliang@zju.edu.cn).

G. Jin is with the College of Information Science and Engineering, Ningbo University, Ningbo 315211, China (e-mail: jinguang@nbu.edu.cn).

Digital Object Identifier 10.1109/LCOMM.2015.2401559

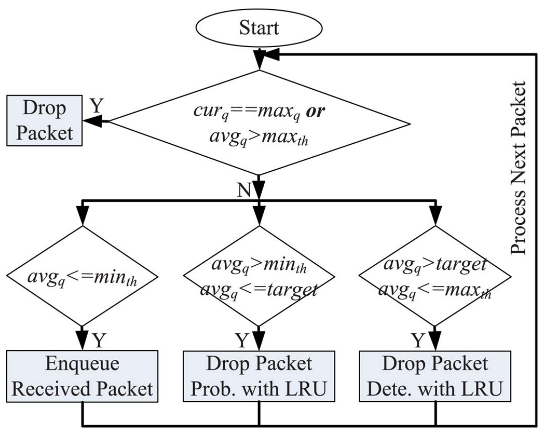
Start
Drop Packet
Y
cur_q==max_q or avg_q>max_{th}
N
avg_q<=min_{th}
avg_q>min_{th} avg_q<=target
avg_q>target avg_q<=max_{th}
Y
Enqueue Received Packet
Drop Packet Prob. with LRU
Drop Packet Dete. with LRU
Process Next Packet

Fig. 1. The queue management process of LRURC.

A. Key Idea and Queue Management of LRURC

LRURC is inspired by the CHOKe algorithm and uses the *sample-match* mechanism to identify high-bandwidth flows. But it employs a *virtual queue* instead of physical queue. To manage partial flows state effectively, LRURC introduces the LRU mechanism. This makes the high-bandwidth flows having high PDP. The key idea of LRURC can be summarized as the following. First, LRURC uses the partial flows state to manage the router's queue (e.g., dropping or marking) when packets arrive at the LRURC-enabled router. Second, according to the arriving packet and packets in the virtual queue, LRURC uses the *sample-match* mechanism and the LRU mechanism to update the partial flows state. Third, LRURC uses the first-in-first-out (FIFO) algorithm to insert the arriving packet's duplicate (or flow's identification) into the virtual queue. Note that the partial flows state are preserved in the LRU cache.

Fig. 1 shows the detailed queue management process of LRURC. Therein, cur_q, avg_q, and max_q are the transient, average, and upper bound of the queue length, respectively. min_{th}, max_{th}, and $target$ are the minimum, maximum, and target threshold of the average queue length (avg_q), respectively. When a packet arrives at the router that has deployed the LRURC scheme, the queue manager uses the records in the LRU cache, cur_q and avg_q to handle the packet. If cur_q equals to max_q, the queue is full and the packet must be dropped. Otherwise, LRURC uses the records in the LRU cache and the average queue length to decide how to handle the packet. If $avg_q \leq min_{th}$, the packet is queued. When $min_{th} < avg_q \leq target$, the packet is dropped probabilistically with p_d that is calculated by

$$p_d = max_p \cdot \frac{f_i}{\sum_{j=1}^{n}(f_j)} + base_p \tag{1}$$

where max_p is the maximum PDP (similar to [1]); f_i and f_j are the frequency of identified as high-bandwidth flow of flow i and j, respectively; $base_p$ is the basic PDP (0.005 in this paper) for holding a stable queue. Note that f_j increases by one when the arriving packet and the duplicate in the virtual queue are matched. If the packet's flow state is not in the LRU cache, its PDP equals to $base_p$. When $target < avg_q \leq max_{th}$, the packet whose flow state exists in the LRU cache is dropped and the other packets is dropped probabilistically with $base_p$. When $avg_q > max_{th}$, all arriving packets are dropped. Note that $target$ equals to $0.5 \cdot (min_{th} + max_{th})$ and avg_q is calculated using the same method in [1].

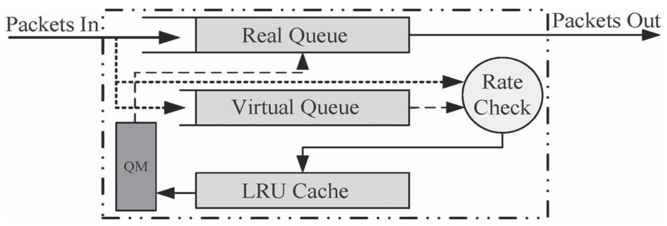

Fig. 2. The structure of LRURC algorithm.

B. Key Components and Implementation of LRURC

As shown in Fig. 2, LRURC has five components including Real Queue (RQ), Virtual Queue (VQ), LRU Cache, Rate Check (RC), and Queue Manager (QM). Therein, RQ is used to store the backlog packets that are not forwarded timely. To identify high-bandwidth flows effectively, LRURC use VQ (like a constant slide window) to preserve some latest historical packets. Note that VQ is always full except the initial stage and its size is restricted to vq_s (one bandwidth-delay product). It makes the identification of high-bandwidth flows more accurate. The LRU cache is used to record several latest high-bandwidth flows and uses the LRU mechanism to manage itself. Note that the size of the LRU cache is restricted to lru_s (100 in this paper). Actually, according to the heavy-tail distribution of network traffic, a small lru_s is sufficient for recording high-bandwidth flows. RC uses Algorithm 1 to identify high-bandwidth flows. QM uses the records in the LRU cache to manage RQ. Its detailed procedure is shown in Fig. 1. Next, we present the implementation of RC.

In RC, we use the *sample-match* mechanism [3], [6] to identify high-bandwidth flows. When a packet arrives at the router's queue, RC uses Algorithm 1 to identify high-bandwidth flows. If the flow identification of the arriving packet is the same as cur_m (similar to [3], [6]) randomly selected packets from VQ, RC returns 1. Otherwise, RC returns 0. Therein, cur_m is less than the maximum matching trial (max_m). $flow\_id$ is the flow identification of the arriving packet. vq_flow_id is the flow identification of randomly selected packets from the *virtual queue*. *cnt* is a counter variable. To compute the packet's flow identification, we adopt a hash function $(FLOW\_HASH(pkt)$ like Line 1 of Algorithm 1). Furthermore, we suppose that the random function is uniform and use a function $(RANDOM\_FETCH(VQ))$ like Line 3 of Algorithm 1 to choose a packet from VQ.

Algorithm 1 Identify High-Bandwidth Flows

```
1: $flow\_id \Leftarrow FLOW\_HASH(pkt)$;
2: for each $i \in [1, cur_m]$ do
3:     $vq\_flow\_id \Leftarrow RANDOM\_FETCH(VQ)$;
4:     if $vq\_flow\_id == flow\_id$
5:         increase $cnt$;
6:     end if
7: end for
8: if $cnt == cur_m$ then
9:     return 1;
10: else
11:     return 0;
12: end if
```

Given that two flows (named h_1 and h_2) passing through the same bottleneck queue and their packets' duplicates share VQ, the packet backlog from h_1 and h_2 are b_1 and b_2, respectively. Suppose that the $RANDOM\_FETCH(VQ)$ function is uniform, according to the analyses of [4], [5], we know that the sampled probability p_i of an incoming flow h_i is $\frac{b_i}{b_1+b_2}$ ($i = 1, 2$). Extended to n flows, the sampled probability of different flows can be calculated by

$$p_k = \frac{b_k}{\sum_{j=1}^{n}(b_j)} \tag{2}$$

where p_k is the sampled probability of the k_{th} flow. After cur_m matches, the hit probability s_k can be calculated as $p_k^{cur_m}$. Obviously, when the flow rate is higher (large b_k), s_k is larger. Therefore, RC returns 1 with higher probability.

When the output of RC equals to 1, the LRU cache must be updated. Like the LRU mechanism, when the flow state is in the LRU cache, its hit counter increases by 1 and the related record is moved to the top of the LRU cache. Otherwise, a new record is created and inserted into the top of the LRU cache. Its hit counter is set to 1.

C. Fairness

One key contribution of LRURC is to protect adaptive flows and guarantee the approximate fairness of various flows. To evaluate the fairness, we adopt the Jain's Fairness Index (JFI) [13] for its popular usage in the AQM (e.g., CoDel [7]). Given that the throughput vector is $(T_1, T_2, \ldots, T_n)$, the JFI can be calculated by (3) which always lies within $[\frac{1}{n}, 1]$. A higher JFI value indicates better fairness among flows.

$$g(T_1, T_2, \ldots, T_n) = \frac{(\sum_{i=1}^{n} T_i)^2}{n\sum_{i=1}^{n} T_i^2}. \tag{3}$$

Suppose that the random function is uniform, the sampled probability of packets in the VQ is proportional with $b_i (i = 1, \cdots, n)$. If b_i is larger, its packet dropping probability will be higher. Hence, the flow rate could be decrease frequently and the fair share of bottleneck queue can be achieved. Given that FA and FC are high-bandwidth non-adaptive flows and FB is a adaptive flow. If the packet backlog of FA and FC are higher than that of FB, the sampled probability of FA and FC will increase as (2). Therefore, the packets of FA and FC are dropped with higher probability as (1). Note that the high-bandwidth flows are recorded in the LRU cache and their PDP is related to the frequency of identified as high-bandwidth flow. Hence, FB is protected. If the packet backlog of FB is higher than others, FB could be restricted by TCP's congestion control mechanism. Therefore, all flows will approach the approximate fairness when the network is stable.

III. Evaluation and Analysis

We use the canonical packet-level simulator NS-2 [14] to conduct a set of experiments to evaluate LRURC in the presence of high-bandwidth non-adaptive flows. RED [1], RED-FT [10], BLUE [2], CHOKe [3], CoDel [7] are compared with LRURC using the JFI [13], standard deviation of queue length and Average FCT (AFCT) [15] metrics. The JFI could reflect the fair share of bottleneck bandwidth, and the AFCT could indirectly reflect queueing delay. The parameter settings of the compared schemes are the optimum value recommended by their respective authors. And the dumbbell and parking-lot topology are widely used in compared schemes.

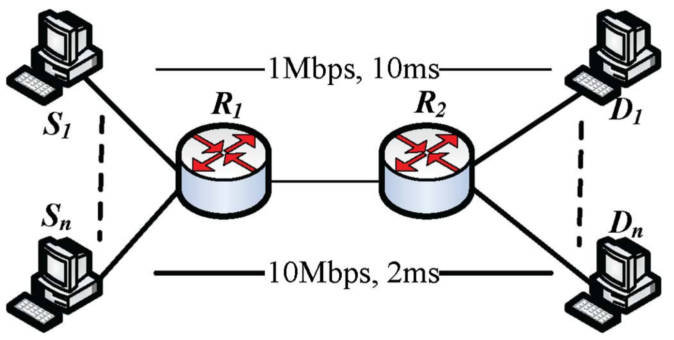

Fig. 3. The dumbbell topology.

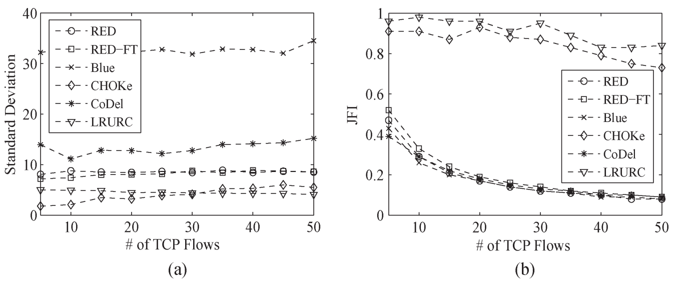

Fig. 4. The fairness and stability of different schemes in dumbbell topology. (a) Queue stability. (b) Fairness index.

Fig. 3 shows the dumbbell topology adopted in this paper. Therein, $R_1 - R_2$ is the bottleneck and its queue size is 100 packets. All the compared schemes are deployed in $R_1 - R_2$ and other queues use DropTail. The delay and capacity of the bottleneck and non-bottleneck (bold line) links are shown in Fig. 3. Note that all flows are from S_i to $D_i (i = 1, \cdots, n)$.

The standard deviation of the queue length is a key metric to reflect the stability of AQM schemes. In this paper, we will use the metric to compare various schemes. The topology of this experiment is shown in Fig. 3. To imitate high-bandwidth flows, two UDP flows, whose sending rate and packet size are 0.5 Mbps and 500 bytes respectively, are established from S_i to D_i (i = 1,2). The number of TCP (long-lived) flows, whose packet size is 1000 bytes, varies from 5 to 50. This can reflect the performance of different schemes more effective. All TCP flows start to transmit data at 0.1 s and stop at 200 s. The UDP flows start at 1.0 s and stop at 200 s. As can be seen from Fig. 4(a), the standard deviation of LRURC is better than other schemes.

The flows' fairness indicates whether AQM schemes can guarantee the fair share of the bottleneck bandwidth among different flows or not when high-bandwidth non-adaptive flows arise. In this paper, we use the JFI metric to analyze the fairness of different schemes. Fig. 3 is the topology of this experiment. The configuration of UDP and TCP flows is the same as before. Note that the throughput for computing the JFI is an average value in [1 s, 200 s]. As can be seen from Fig. 4(b), LRURC is better than other schemes and can guarantee the fair sharing when high-bandwidth non-adaptive flows arise.

Regarding the AFCT, we use short-lived TCP flows, whose packet size is 1000 bytes, to measure it. The flow size is a uniform distribution and its mean value varies from 10 to 100 packets. The number of short-lived flows varies from 100 to

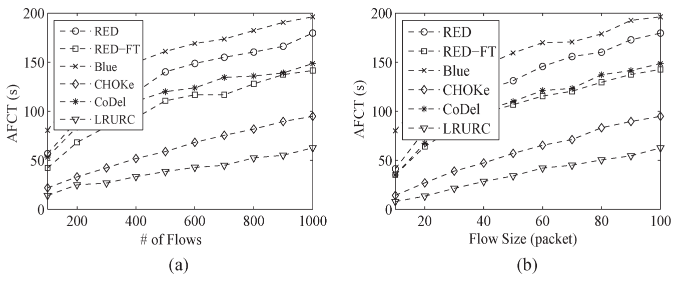

Fig. 5. The AFCT of different schemes in dumbbell topology. (a) Flow Num. vs. AFCT. (b) Flow size vs. AFCT.

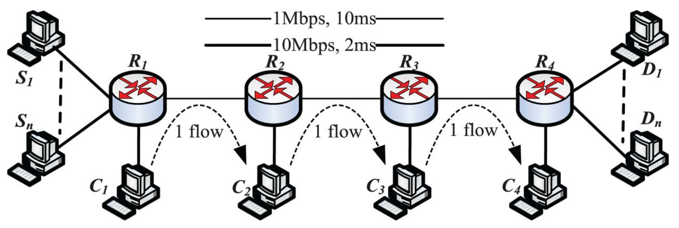

Fig. 6. The parking-lot topology.

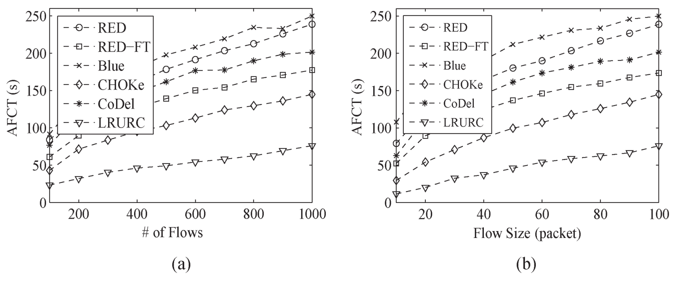

Fig. 7. The AFCT of different schemes in parking-lot topology. (a) Flow Num. vs. AFCT. (b) Flow size vs. AFCT.

1000. Two UDP flows is the same as before. At 0 s, the short-lived TCP flows start to transmit data. Note that the simulation time is un-limited. The results are shown in Fig. 5, and we can see that LRURC outperforms other schemes as far as the AFCT is concerned.

We also use a parking-lot topology (as Fig. 6) to evaluate the performance of LRURC. Therein, $R_1 - R_2$, $R_2 - R_3$ and $R_3 - R_4$ are the bottleneck and their queue size is 100 packets. All compared schemes are deployed in the bottleneck and other queues use DropTail. The delay and capacity of the bottleneck and non-bottleneck (bold line) links are shown in Fig. 6. Three UDP flows, whose sending rate and packet size are 0.5Mbps and 500 bytes respectively, are established from C_i to C_{i+1} ($i = 1,2,3$). Due to page limited, only the AFCT results are listed in this paper. Similarly, we also use the same model to build the short-lived TCP flows as before for measuring the AFCT. Note that all short-lived flows are from S_i to $D_i (i = 1, \cdots, n)$ and the packet size is 1000 bytes. The results are shown in Fig. 7, and we can see that LRURC outperforms other schemes as far as the AFCT is concerned.

In summary, LRURC can achieve higher JFI as well as lower AFCT. Furthermore, it can hold a stable queue whose queue length is around an expected value. In terms of space efficiency, LRURC consumes some additional memory (e.g., VQ and LRU cache). But they are a constant. Due to the simplicity and efficiency, LRURC can be easily deployed in current Internet routers.

IV. Conclusion

In this letter, we have presented a novel AQM algorithm (named LRURC) based on the *sample-match* (like [3]), virtual queue, and LRU mechanism. It can achieve approximate fairness among various flows, reduce the flow completion time, and hold the queue length around the expected value. Furthermore, the scheme is easy to deploy in current Internet routers. Further work will focus on the complexity and stability analysis, the adaptability and optimization of the parameters in more complex scenarios etc.

References

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

[2] W. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "BLUE active queue management algorithms," *IEEE/ACM Trans. Netw.*, vol. 10, no. 8, pp. 513–528, Aug. 2002.

[3] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe: A stateless active queue management scheme for approximating fair bandwidth allocation," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 942–951.

[4] A. Tang, J. Wang, and S. H. Low, "Understanding CHOKe: Throughput and spatial characteristics," *IEEE/ACM Trans. Netw.*, vol. 12, no. 4, pp. 694–707, Apr. 2004.

[5] A. T. Eshete and Y. Jiang, "On the transient behavior of CHOKe," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 875–888, Mar. 2014.

[6] A. Eshete and Y. Jiang, "Generalizing the CHOKe flow protection," *Comput. Netw.*, vol. 57, no. 1, pp. 147–161, Jan. 2013.

[7] K. Nichols and V. Jacobson, "Controlling queue delay," *ACM Queue*, vol. 10, no. 5, pp. 1–15, May 2012.

[8] T. H. Joergensen *et al.*, "FlowQueue-Codel," Mar. 2014. [Online]. Available: https://tools.ietf.org/html/draft-hoeiland-joergensen-aqm-fq-codel-00

[9] C. Zhang, J. Yin, Z. Cai, and W. Chen, "RRED: Robust RED algorithm to counterlow-rate denial-of-service attacks," *IEEE Commun. Lett.*, vol. 14, no. 5, pp. 489–491, May 2010.

[10] X. Jiang, J. Yang, G. Jin, and W. Wei, "RED-FT: A scalable random early detection scheme with flow trust against DoS attacks," *IEEE Commun. Lett.*, vol. 17, no. 5, pp. 1032–1035, May 2013.

[11] F. Lu, G. M. Voelker, and A. C. Snoeren, "Weighted fair queuing with differential dropping," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2981–2985.

[12] R. Adams, "Active queue management: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1425–1476, 3rd Quart. 2013.

[13] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, Jun. 1989.

[14] D. Chiu and R. Jain, "The Network Simulator Version 2," 2012. [Online]. Available: http://www.isi.edu/nsnam/ns/

[15] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, Jan. 2006.