Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 628583, 13 pages
http://dx.doi.org/10.1155/2015/628583

Research Article

Cascade Probability Control to Mitigate Bufferbloat under Multiple Real-World TCP Stacks

Hoang-Linh To,[1] Thuyen Minh Thi,[1] and Won-Joo Hwang[2]

[1]*Department of Information and Communication System, Inje University, Gimhae, Gyeongnam 621-749, Republic of Korea*
[2]*Department of Information and Communications Engineering, Inje University, Gimhae, Gyeongnam 621-749, Republic of Korea*

Correspondence should be addressed to Won-Joo Hwang; ichwang@inje.ac.kr

Received 22 June 2015; Accepted 13 August 2015

Academic Editor: Rongwei Guo

Persistently full buffer problem, commonly known as bufferbloat, causes unnecessary additional latency and throughput degradation whenever congestion happens in Internet. Several proposed queue management schemes, with the debloat mission, are almost based on the modification of one-loop feedback control where the instability and bad transient behavior are still big challenges. In this paper, we present a cascade probability control scheme using margin optimal method to address such challenges under different kinds of real-world TCP stacks. Simulation results guarantee the measured round trip time tracking to a low value of delay (e.g., ≈180 ms under TCP Reno, and ≈130 ms under TCP Cubic) and ≈50% delay reduction in comparison to current deployed queue management schemes in network devices.

1. Introduction

Nowadays, interactive and delay-sensitive applications such as VoIP, teleconference, and gaming often perform more poorly than before. From the economic perspective, as the cost of memory has decreased over the past few years, memory with large capacity has been put into network devices such as routers. From the engineering perspective, as traditional analysis, a larger buffer results in less loss rate under congestion. However, recent studies about *bufferbloat* reveal some bad effects of a large buffer. It might destroy interactivity of transport control protocols (TCP) under load and often results in higher latency and lower throughput. Nagle [1] firstly drew attention to effects of infinite buffers in packet-switching networks. Then *bufferbloat* termed by Gettys [2] firstly opens a research field to seriously reconsider the problem of large buffers and hidden "dark" buffers which can appear everywhere in Internet devices.

Conceptually, bufferbloat is a phenomenon firstly realized in packet-switched networks, where excess buffering of packets might cause high latency and jitters, as well as reducing the overall network throughput. When a router device is equipped with a large buffer, it can become practically unusable for many delay-sensitive applications like voice calls, chat, or even web surfing. Traditional rule-of-thumb for setting buffer size in an Internet route is based on bandwidth-delay product (BDP), the product of a data link's capacity and its round-trip time (RTT). The basic premise is that modern networking kit contains too much buffer memory. In [2], it is necessary to restrict bandwidth to improve latency and ping times. Despite this fundamental of sacrificing bandwidth for latency, most users keep asking how to fill their bandwidth quota and get good VoIP/games satisfaction. Long time ago and until now, researchers nonexhaustively attempt to reduce dropping packets by adding more and more buffering in routers. However, each dropped packet is essential for well-behaved functioning of TCP in congestion.

Let us examine an interaction situation between TCP and bufferbloat [3]. TCP relies on timely congestion notification to adjust its transmission rate to the available bandwidth. And bufferbloat means that new arriving packets are continued to be buffered, instead of dropped due to large buffer size. It causes the queue to build up longer at the bottlenecks. Entering the dropping state, more packets are dropped than necessary which would shrink the transmission rates of the TCP sender. In particular, if several TCP applications are

transmitting over the same congestion point, all flows will see drops at the same time. Therefore, all transmission rates would be reduced simultaneously, called TCP global synchronization. Certainly, suitable amount of buffering is helpful to improve the efficiency of data transmissions to smoothen bursts transmissions. Not dropping packets early enough, however, leads to increasing delays for interactive real-time communication since widely deployed congestion control algorithms only rely on packet loss as a signal for congestion. Not dropping packets in a timely fashion also prevents TCP applications from reacting properly to overload.

Solutions to bufferbloat problem or debloat include queue management algorithms, which attempt to manage queue occupancy in passive or active ways. Passive queue management (PQM) such as Drop Tail is the most currently deployed queue management algorithm in Internet router devices. It drops packets when buffer queue is overflow, so because of this simplicity, it is widely used now. The weaknesses of PQM such as lock-out [4] and global synchronization [5] motivate to urgent needs of active queue management schemes (AQMs) deployment. Several AQMs have been deeply studied in the recent decade. The most popular one is random early detection (RED) [5] and its variants [6, 7] with the main idea of using a probabilistic approach to randomly dropping packets for congestion resolution. However, most of them require careful tuning for varying network conditions; otherwise they only work well under a few specific scenarios with default parameters [8]. Game theory approach, which is mainly dedicated to users (user-aware), has also been investigated to tackle this issue [9, 10]. Recently, a new AQM named controlled delay management (CoDel) [11] has been proposed to overcome weakness. Parameterless and easy deployment are two among strong points of CoDel. Even though, for larger RTTs and smaller bandwidths, CoDel has poor link utilization than RED and its variants, the next drop time of CoDel algorithm is derived by using the multiplicative decrease of square root of total number dropped packet counting, which needs more in-depth investigation for improvement.

In this paper, we revisit AQM design problem from control theoretical perspective but consider cascade control to further optimize performance in user application level while considering bufferbloat phenomenon. Control theory is one of the most efficient tools for AQM to bring a better system stability due to well-developed control theory background. Several efforts have been recently put into this direction for RED [12], PID-AQM [13, 14], and controller design in state space [15, 16] or queuing modelling with the impatient customer feature [17] and so forth. Based on the fluid-flow approximation model for additive increase-multiplicative decrease (AIMD) phase of TCP, they converted dynamic equations of window size and queue length into system transfer function or state space by linearization methods. Then they designed the specific controller according to a closed-loop transfer function of the whole TCP system. We realize that these models almost design one-loop control for queue length only which creates some problem of difficult adjusting controller parameters and stability guarantee. One more loop with cascade design in control theory would improve their performance, especially the bad transient behaviour of current debloat schemes. So we consider a two-loop control scheme for AQM, which is mainly dedicated to bufferbloat issue. Dividing into an inner and an outer loop, the inner one adjusts window size based on changing of traffic and feedback window size at time t. The outer one's mission is to adjust queue length based on feedback queue length value at time t. Each inner loop and outer loop are designed using two transfer functions which decomposed from the fluid-flow model and, therefore, have different controllers. One difficulty when considering this cascade design is the interaction in time-scale between two loops. We see that the inner loop operates in the transport layer while the outer one operates in link layer (faster than transport layer). The main motivation is that solving bufferbloat (large buffer) problem cannot be done without considering different network layers simultaneously. Therefore, one-loop control methods like Drop Tail, RED, or classical PID should not be adequate for bufferbloat mitigation. Our proposed CPC, with an inner and an outer loop, acts in both the transport layer (adjusting window size) and the link layer (adjusting queue length). To the best of our knowledge, our work is the first attempt to adapt cascade control method to bufferbloat research field. The weakness of this method is that more complexity is added because of two additional controllers. However, better performance results (shorter queuing delay and larger goodput) are achieved and we are going to see more details in Section 5. The performance metrics we evaluate consist of measured round-trip time, queue length at each instantaneous time, and goodput at TCP application layer which represents the users' (clients) satisfaction level. The primary contributions are summarized as follows:

(i) We propose a cascade probability control (CPC) which has two control loops. The inner loop gets information from current window size and capacity of link while the outer loop is based on difference between average measured queue length and queue length reference value.

(ii) Section 4 presents our proposed CPC controllers which are designed using the optimal margin method in frequency domain. The fast transient and stability in a wide frequency range can be achieved using this method.

(iii) We develop our own model to simulate CPC performance using the popular open-source software, OMNeT++. We also compare our proposal with Drop Tail and RED under three popular TCP stacks (e.g., Reno, Cubic, and FreeBSD.) using the real-world stack module network simulation cradle (NSC). Simulation results demonstrate that bufferbloat delay can be reduced significantly (≈50% compared to Drop Tail and RED) and well controlled by CPC (Section 5).

2. Related Works

Several approaches to AQM, using control theory with the core linearized TCP model by Hollot et al. [12], were proposed

and contributed a large portion to debloat field [18]. The original goal was to propose more concrete design guidelines for the RED parameters to improve stability and responsiveness; however, they also discover clearer understanding of RED's behaviour with changing of network conditions such as round-trip times, offer load as number of TCP flows, and link capacities. Focusing on control theory viewpoint, we categorize them into three types: classical control, robust control, and fuzzy logic control.

Classical PID controller based algorithms were designed as alternative AQM solutions to meet various specifications of the Internet using feedback control. Hollot et al. [12] analysed RED as I-controller and proposed two types, the proportional (P) controller and the proportional integral (PI) controller, for improving RED. The stable region of the control gain is determined for dynamic-RED (DRED) using the Routh stability test. In [19], dynamic-RED (DRED) was proposed, using a load-dependent probability to randomly discard packets when a buffer becomes congested. It maintains the average queue size close to a predetermined threshold but allows transient traffic bursts to be queued without unnecessary packet drops. Its main advantage is that we do not have to collect state information of individual flows. In [15], a feedback control model with PI controller has been recently proposed to improve link performance in wired communication networks.

Robust control approach was also studied to improve classical control. The issue of large delay with large buffer in bufferbloat was addressed by DC-AQM algorithm based on internal mode compensation (IMC) principle, which is an example of robust control approach [20]. Using IMC controller derived, they tried to tune parameters K_p, K_i, and K_d of classical PID controller to reduce delay. To also contend with large delay, gain adaptive Smith predictor with PI controller (GAS-PI) in [21] was built to improve robustness. Then, in [13], a predictive PID controller is proposed for TCP/AQM. They used the generalized predictive control method to determine suitable values for K_p, K_i, and K_d so that they made the system more robust to changes in model parameters such as offered load and round-trip time.

Fuzzy control RED (FCRED) was proposed in [6]. It consists of a fuzzy controller adjusting the $P_{\max}$ parameter of the RED algorithm. The fuzzy controller includes three parts: the fuzzification unit followed by the fuzzy-interference engine with fuzzy-rule base and finally a defuzzification unit. The fuzzification module maps the input values to be controlled to a fuzzy set (i.e., membership functions). The fuzzy rule base provides the connection between the input signals and the appropriate output variable. Fuzzy logic rules are constructed based on trial-and-error, which needs the knowledge and experience of domain-expert in TCP congestion control. Moreover, fuzzy logic-based AQM schemes are sometimes not distributed and hard to implement.

Briefly, it can be seen that, as the years progress, the main direction has been to more and more sophisticated robust control techniques, combined with some classical techniques in control theory as well. In many cases, the linear TCP model in [12] continues to be at the core of control theoretic AQM algorithms. Cascade control, however, has not been considered yet in development of AQM. In this paper, we contribute to the AQM debloat research trends by using cascade control to address large delay bufferbloat issue. Our approach uses two controllers which are connected in cascade style. Our CPC results obtained from theoretical Matlab or simulation using OMNeT++ and the integrated NSC tool are so promising in fast transient behavior and stability.

3. System Model

The TCP/AQM fluid-flow model described by nonlinear differential equations has been extensively studied in particular AQM routers interacting with TCP sources (e.g., [12, 13, 16, 20]). Until now, they can capture the additive increase-multiplicative decrease (AIMD) feature from TCP [22], without slow start and time-out mechanisms. However, this lacking only affects initial start-up of the system. Once the system reaches the stable point, the differential equations solver is able to track changes in the network well [23]:

$$\begin{aligned} \dot{w}(t) &= \frac{1}{R(t)} - \frac{w(t)}{2}\frac{w(t-\tau)}{R(t-\tau)}p(t-\tau); \\ \dot{q}(t) &= N\frac{w(t)}{R(t)} - C_l, \end{aligned} \tag{1}$$

where

(i) C_l is the transmission capacity of link l (packets/sec);

(ii) R is the round-trip time (sec); $R(t) = T_p + q(t)/C_l$ with T_p being the fixed propagation delay;

(iii) N is the number of TCP flows;

(iv) τ is feedback delay (sec).

The operating point (w_0, q_0, p_0) of TCP model (1) can be derived at $\dot{w} = 0$ and $\dot{q} = 0$ as follows:

$$\begin{aligned} w_0^2 p_0 &= 2, \\ w_0 &= \frac{R_0 C_l}{N}, \\ R_0 &= T_p + \frac{q_0}{C_l}. \end{aligned} \tag{2}$$

Doing linearization of the above TCP queue model around operating point (w_0, q_0, p_0) (details in [12]), with $\delta\dot{w} = w - w_0$, we have a linearized small signal model of TCP/AQM:

$$\begin{aligned} \delta\dot{w}(t) &= -\delta w(t)\frac{2N}{R_0^2 C_l} - \delta p(t - R_0)\frac{R_0 C_l^2}{2N^2}, \\ \delta\dot{q}(t) &= \delta w(t)\frac{N}{R_0} - \delta q(t)\frac{1}{R_0}. \end{aligned} \tag{3}$$

Convert (3) using Laplace transformation as follows:

$$\begin{aligned} s \times w(s) &= -w(s)\frac{2N}{R_0^2 C_l} - p(s)e^{-sR_0} \cdot \frac{R_0 C_l^2}{2N^2}, \\ s \times q(s) &= w(s)\frac{N}{R_0} - q(s)\frac{1}{R_0}. \end{aligned} \tag{4}$$

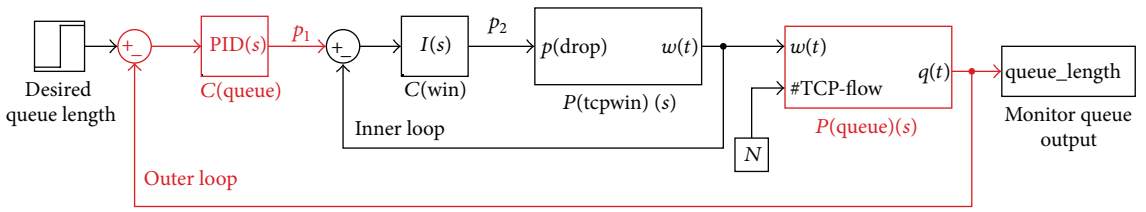


Figure 1: Block diagram: proposed cascade probability control scheme.

So the TCP/AQM system transfer functions in Laplace domain are

$$P(s) = P_{\text{tcpwin}}(s) \cdot P_{\text{queue}}(s) \cdot e^{-sR_0} = \left[\frac{w(s)}{p(s)}\right] \cdot \left[\frac{q(s)}{w(s)}\right] = \left[\frac{A}{s+B}\right] \cdot \left[\frac{C}{s+D}\right] e^{-sR_0}, \tag{5}$$

where $A = R_0C_l^2/(2N^2)$; $B = 2N/(R_0^2C_l)$; $C = N/R_0$; and $D = 1/R_0$.

4. Cascade Probability Control (CPC)

In literature, single loop control for AQM was often studied due to less complexity and low oscillation but did not behave well in case of disturbances (e.g., bandwidth fluctuation and bursty traffic) or bad transient. Cascade control comes to rescue and achieves fast rejection of disturbance before it affects the main system model. In this section, we propose a cascade probability control (CPC) to improve dynamic of the open-loop system transfer function (5) which consists of two subsystems, window size and queue length control. The inner loop adjusts the window size by an inner controller C_{win}, while the outer loop receives queue length information at time t and uses an outer controller C_{queue} to maintain $q(t)$ (Figure 1). The final goal is to reach a stable value of queue length, so that bufferbloat phenomenon can be mitigated. Moreover, the margin optimal method for controller design also brings a fast transient behavior to our system.

4.1. CPC Controllers. We design two PI controllers for simplicity and for reducing number of parameters: the inner controls dropping probability p_2 based on traffic information and the outer controls dropping probability p_1 based on difference between measured average queue length and reference queue length. We outline our proposed design framework in Figure 1 and would compete with the adaptive weight PID approach in [24].

4.1.1. Inner Loop. An important design requirement is that the inner loop controller should behave quickly. From (5), the inner control objective is a linear first-order type: $P_{\text{tcpwin}}(s) = k_I/(1+Ts)$, where $k_I = A/B$ and $T = 1/B$. Hence, we design an integral I-controller for inner loop:

$$C_{\text{win}(s)} = \frac{1}{T_{C_{\text{win}}}s}. \tag{6}$$

(i) The close-loop transfer function of the inner loop is

$$I(s) = \frac{w(s)}{p_1(s)} = \frac{P_{\text{tcpwin}}(s)\,C_{\text{win}(s)}}{1 + P_{\text{tcpwin}}(s)\,C_{\text{win}(s)}} = \frac{k_I}{T_{C_{\text{win}}}s(1+Ts) + k_I}. \tag{7}$$

(ii) The $I(s)$ transfer function is converted into frequency domain, with ω being frequency:

$$|I(j\omega)| = \frac{k_I}{\sqrt{\left(k_I - T_{C_{\text{win}}}T \cdot \omega^2\right)^2 + \left(\omega \cdot T_{C_{\text{win}}}\right)^2}}, \quad \Longleftrightarrow I(j\omega)^2 = \frac{k_I^2}{k_I^2 + \left(T_{C_{\text{win}}}^2 - 2k_I \cdot T_{C_{\text{win}}}T\right)\omega^2 + T_{C_{\text{win}}}^2 \cdot T^2\omega^4}. \tag{8}$$

One of quality requirements to closed-loop control system, which is represented by $I(s)$, is that the output is the same as the input signal or the controller C_{win} should bring $|I(j\omega)| = 1$, $\forall\omega$, which can be called margin optimal method. However, due to several reasons of real system, that requirement is rarely satisfied for all frequencies ω. An acceptable design is that $|I(j\omega)| \approx 1$, in a wide band of low frequencies ω. Hence, we propose choosing $T_{C_{\text{win}}}$ such that $T_{C_{\text{win}}}^2 - 2k_I \cdot T_{C_{\text{win}}}T = 0$ or $T_{C_{\text{win}}} = 2k_IT$. This close-form expression of $T_{C_{\text{win}}}$ is used to make decision for controller C_{win}.

4.1.2. Outer Loop

(i) The close-loop transfer function of the outer loop is

$$O(s) = \frac{q(s)}{q_{\text{ref}}} = \frac{I(s)\,P_{\text{queue}(s)}C_{\text{queue}}(s)}{1 + I(s)\,P_{\text{queue}(s)}C_{\text{queue}}(s)}. \tag{9}$$

(ii) The outer control objective into zero-pole form is

$$I(s)\,P_{\text{queue}(s)} = \frac{k_I}{T_{C_{\text{win}}}s(1+Ts) + k_I} \cdot \frac{C}{s+D} = \frac{k_O}{\left(1+T_{1_O}s\right)\left(1+T_{2_O}s\right)\left(1+T_{3_O}s\right)}, \tag{10}$$

where $k_O = C/D$; $T_{1_O}T_{2_O} = TT_{C_{\text{win}}}/k_I$; $T_{1_O} + T_{2_O} = T_{C_{\text{win}}}/k_I$; $T_{3_O} = 1/D$.

For the outer loop, the objective function is linear third-order type, due to inclusion of $I(s)$. Hence, we choose proportional-integral-derivative PID controller by using the same method at the inner loop design, or $|O(j\omega)| \approx 1$:

$$C_{\text{queue}(s)} = k_{p_O}\left(1 + \frac{1}{T_{i_O}s} + T_{d_O}s\right), \tag{11}$$

with $k_{p_O} = (T_{1_O} + T_{2_O})/2k_OT_{3_O}$, $T_{i_O} = T_{1_O} + T_{2_O}$, $T_{d_O} = (T_{1_O} \cdot T_{2_O})/(T_{1_O} + T_{2_O})$.

4.2. CPC Numerical Analysis. The proposed CPC controllers can be easily verified by using example parameters from [12] which consist of $N = 60$, $C_l = 3750$ (packets/sec), and $R_0 = 0.246$ (sec). According to the above analysis, the inner controller $C_{\text{win}} = 0.000291$/s, while the outer controller $C_{\text{queue}} = 0.1281(1 + 1/(3.78\,\text{s}) + 1.89\,\text{s})$.

Figure 2(a) presents the queue length output in case of 20 packets queue reference. The CPC manual tune mode uses our above designed controllers, while the CPC autotune mode uses *pidtune* function of Matlab. Firstly, in comparison to RED which can be modeled as a single-loop I-type controller, one advantage of CPC is that fast transient behavior can be achieved. Transient behavior is a major issue of current solutions to bufferbloat. Fast transient means that we can reach the queue length reference quickly in response to the input change of dynamic systems.

Secondly, crossover frequency is a criterion to assess a control system's operation ability in a wide range of frequency. The higher the crossover frequency is, the better stability at which the system can operate is. Let us denote crossover frequency as ω_c. If we choose a frequency higher than ω_c, system would be not stable anymore. The left-hand side of Figure 2(b) is Bode diagram phase-margin of the system. It informs about ω_c information of our proposed CPC scheme. Specifically, CPC manual tune has $\omega_c = 2.37$ (rad/s) which is the highest value, while $\omega_c = 0.627$ (rad/s) for CPC autotune and $\omega_c = 0.261$ (rad/s) for RED. Therefore, CPC can operate better in a wider range of frequency.

Finally, using Nyquist stability criterion, the closed-loop transfer function of the outer loop $O(s)$ is determined by the values of its poles. It states that, for stability, the real part of all poles must be negative or the poles are in the left half-plane of pole-zero map. The right-hand side of Figure 2(b) shows us the distribution of zeros and poles for three schemes. Clearly, all the poles have the real part negative which strongly demonstrates CPC's stability.

In summary, CPC scheme can achieve fast transient, high crossover frequency and still stable. Those motivate us to conduct simulations about CPC behavior to bufferbloat under multiple real-world TCP stacks.

5. Simulation Results

In this section, we implement the proposed scheme in simulator to show bufferbloat phenomenon and the advantages of CPC scheme in comparison to traditional AQMs. We choose the dumbbell topology according to AQM guidelines [25].

Table 1: Simulation parameters.

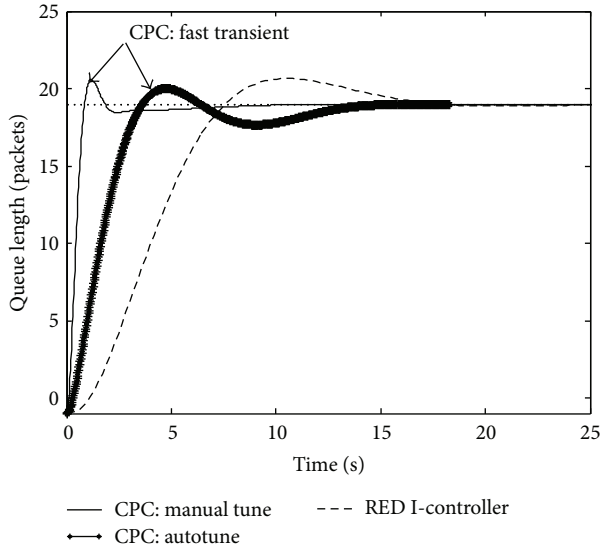
Name	Value	Unit
Maximum buffer size	500	Packets
Desired queue length	20	Packets
Target RTT delay	200	ms

5.1. Simulation Setup. We develop our own simulation model to verify the proposed CPC scheme using the popular OMNeT++ framework [26]. The chosen topology in Figure 3 represents a dumbbell network which is suitable for evaluating a queue management scheme. Three clients simultaneously send 200×3 (MByte) data to servers through intermediate routers. The advertised window's server is set up to infinity so that it does not limit the sender's speed. To create an artificial bottleneck, we set up the "high-to-slow" link bandwidth where the bandwidth of links from clients to router 1 are high at 1000 (Mbps) and the bandwidth of the link from router 1 to others is slow at 2 (Mbps). RTT can be measured at each server, while queue size is monitored at the congestion point. Bursty traffic is generated using a generator traffic model inside each client from OMNeT++. At the first glance, we vary the buffer size parameter to show the simulation model working toward bufferbloat phenomenon existence. Then we compare the CPC scheme with two types of traditional queue discipline which are operating inside the current Internet, Drop Tail, and RED, in terms of queue size and round-trip time or latency. Some main parameters are summarized in Table 1.

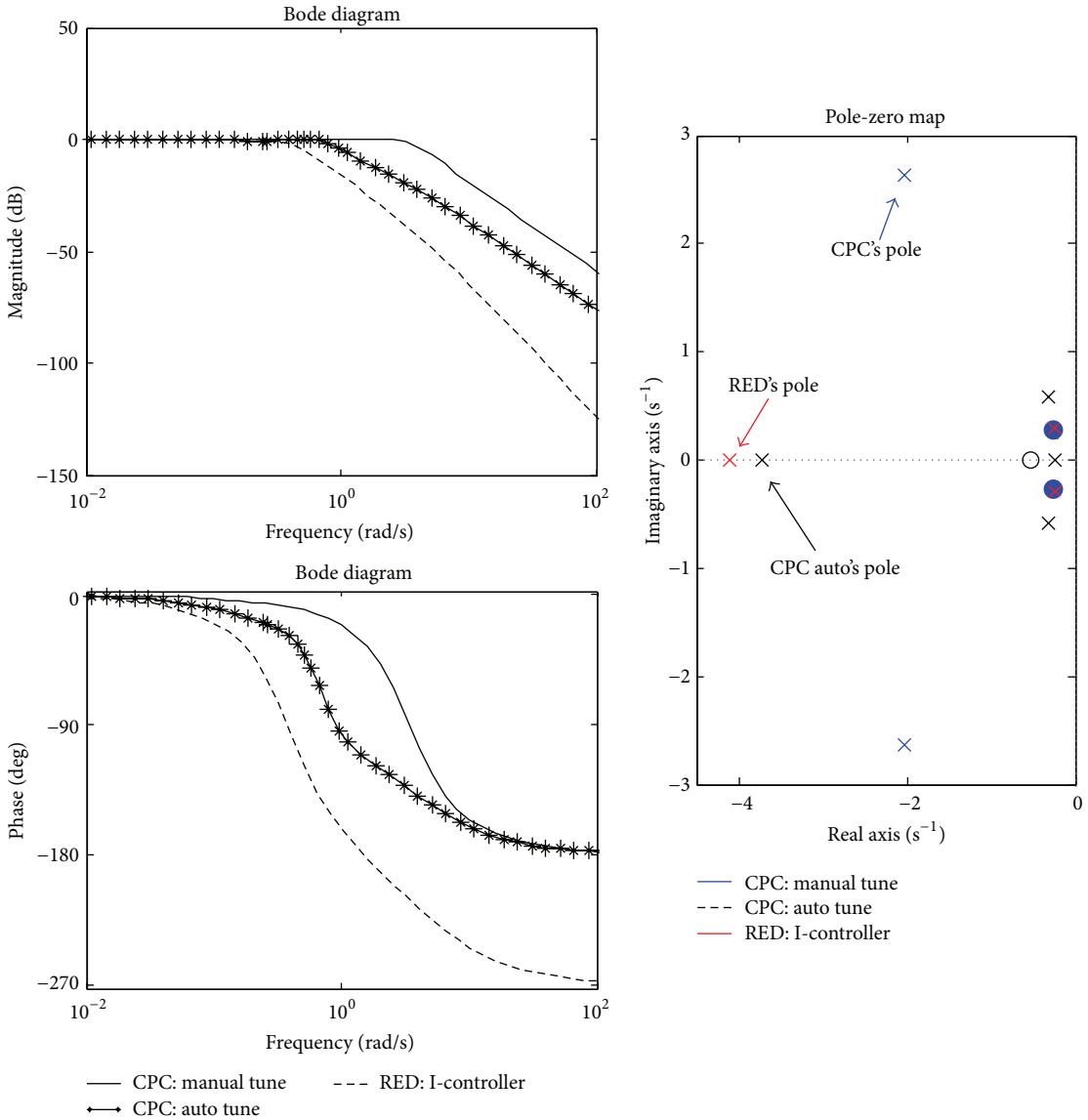
5.2. Bufferbloat Existence

5.2.1. Experiment. To demonstrate clearly bufferbloat existence, we exploit the real-time response under load (RRUL) test specification in [27]. This test puts a network under worst case conditions and then measures latency responsiveness and other relative performances of TCP and UDP streams of varying rates, while under that load. Then, Toke Hoiland-Jorgensen produces a wrapper for netperf tool to implement test cases such as HTTP, VoIP, and FTP under RRUL. In this paper, we conduct the experiment test from our computer-client located in South Korea to two servers which are mainly dedicated for bufferbloat testing, demo.tohojo.dk, and snapon.lab.bufferbloat.net, respectively. Figures 4(a) and 4(b) present our ping results when we run experiment in 300 seconds and sampling period of 0.2 second. Ping delay for both cases are approximately 300 (ms), while $100 \rightarrow 150$ (ms) is an acceptable range for toll-quality voice and delay-sensitive applications. Once again, we see that debloat solutions were not deployed in our current experiment routers.

5.2.2. Simulation. As mentioned before, the main reason for bufferbloat is the unmanaged large buffer at bottleneck



(a) Step response performance



(b) Bode diagram and pole-zero map

Figure 2: Stability analysis of CPC.

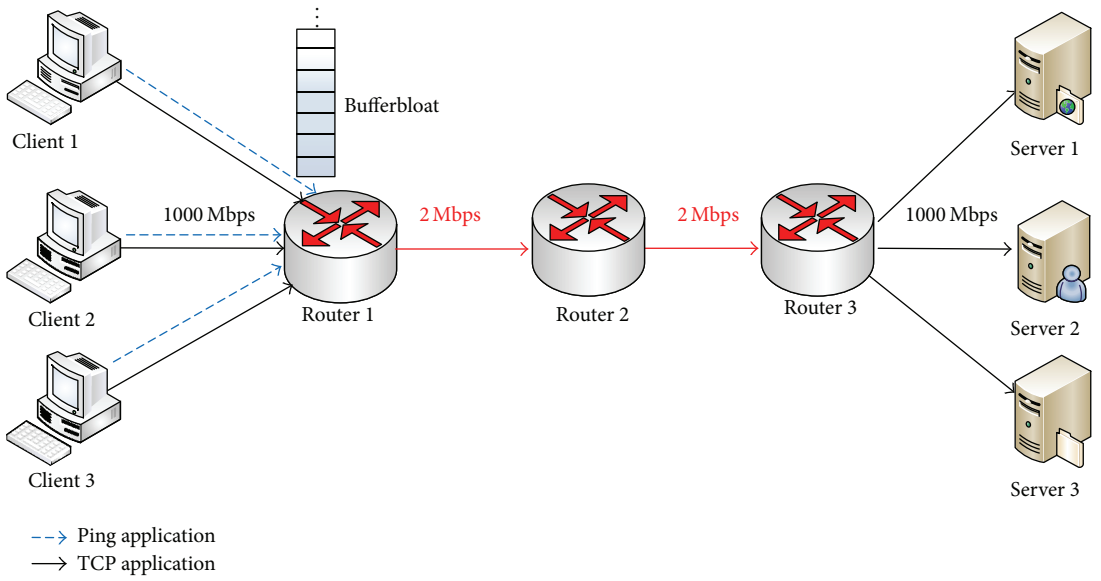
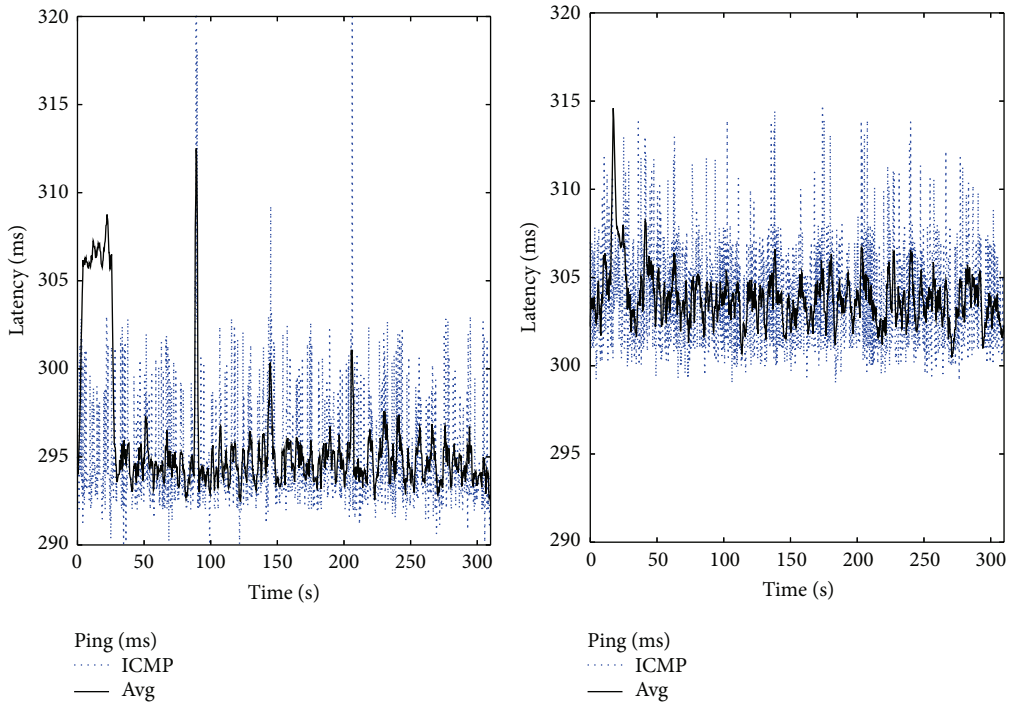


Figure 3: Simulation topology.



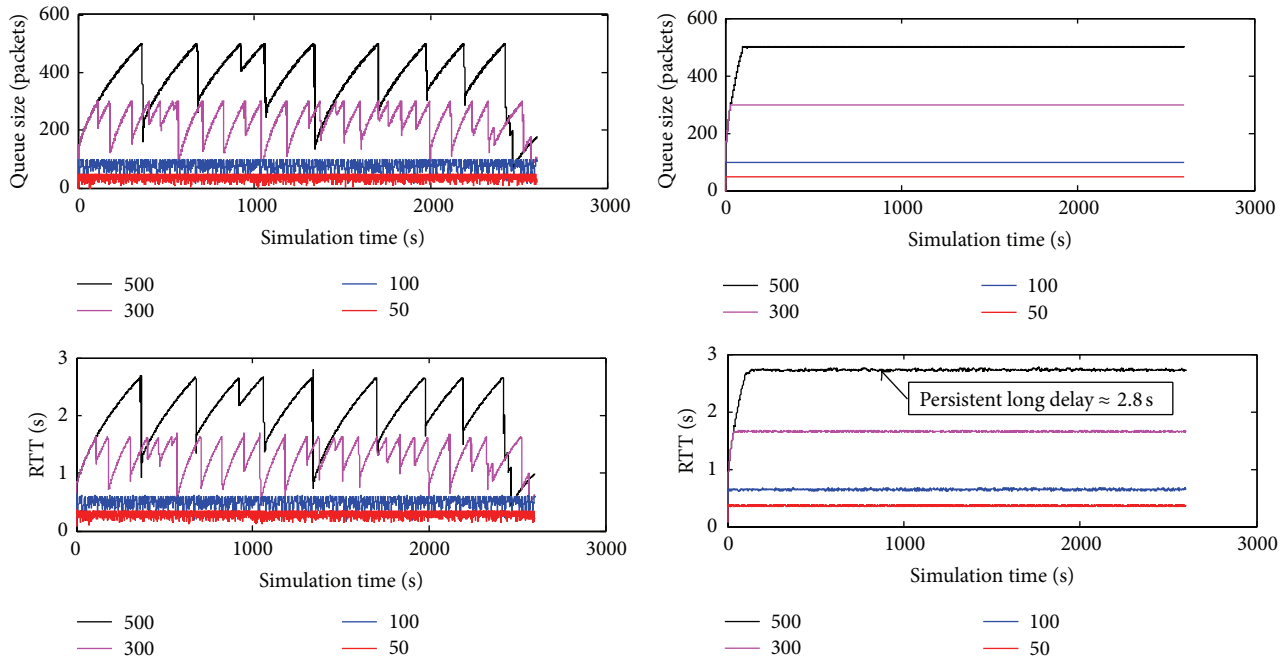
(a) Server: demo.tohojo.dk

(b) Server: snapon.lab.bufferbloat.dk

Figure 4: Bufferbloat existence: an experiment from Korea to two servers.

links that could damage TCP's congestion control/avoidance mechanisms. Hence, a big buffer is necessary to reproduce this phenomenon. In the current Internet, buffers are measured using bytes. OMNeT++ simulation framework, however, uses packets to sizing buffers. The choice of the largest buffer size is approximately 8x BDP product (packets) [11]. Let us denote buffer size as bu. With bottleneck bandwidth 2 Mbps and propagation delay 300 ms, we have the results when following the traditional rule-of-thumb for buffer-size:

$$\begin{aligned} \text{bu}\,(\text{byte}) = \text{BDP} = \text{BW} \times \text{Delay} = 2.10^{6} \times 300.10^{-3} \\ = 600\ (\text{kbit}) = 75\ (\text{kbyte})\,. \end{aligned} \tag{12}$$



(a) TCP Reno + Drop Tail

(b) TCP Free BSD + Drop Tail

Figure 5: Bufferbloat existence and varied buffer sizes.

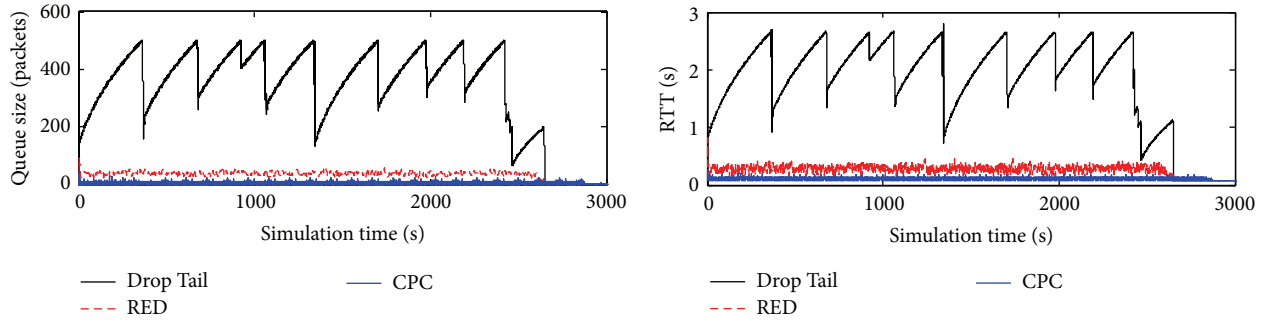
According to [28], one TCP packet size or maximum transmission unit (MTU) is 1500 (bytes):

$$\text{bu}\,(\text{packets}) = \frac{\text{bu}\,(\text{byte})}{\text{MTU}} = \frac{75.10^3}{1500} = 50\,(\text{packets})\,. \quad (13)$$

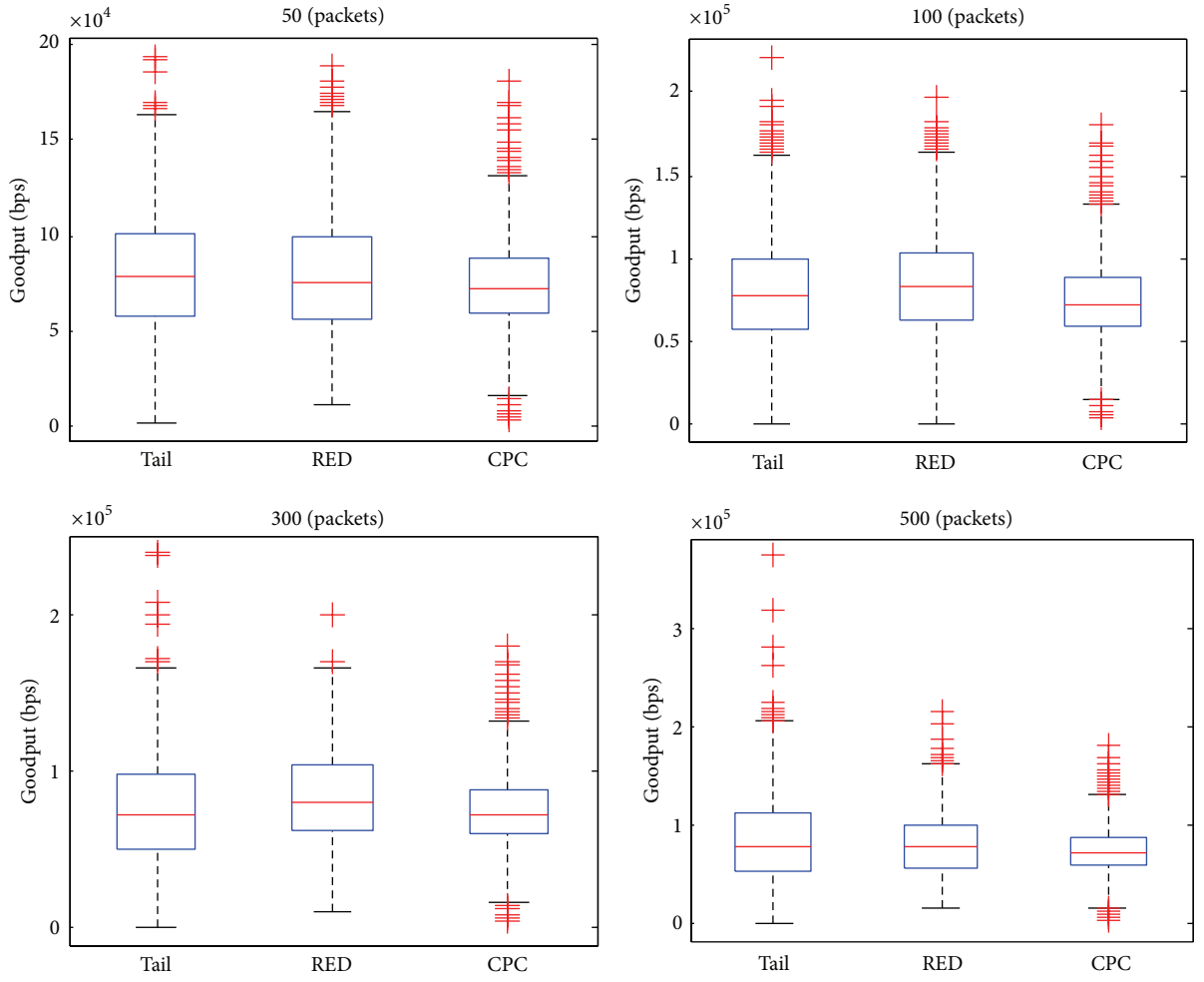
Therefore, in this simulation, we vary the buffer sizes such as 50, 100, 300, and 500 being the largest one (packets) under different TCP congestion-avoidance algorithms with Drop Tail queuing discipline. Firstly, Figure 5(a) shows the effects on queue evolution and round-trip time, under TCP Reno, the most used in Windows and some Linux operating systems (OS). We obtain the saw-tooth result at both queue size and RTT graph because of TCP Reno's characteristics while others do not have saw-tooth type graphs. From Figure 5(a), bufferbloat issue can also be clearly recognized. If buffer size is very large (500 packets), RTT will become pretty large (≈2.5 sec) while queue size grows until it reaches the threshold of buffer, which is the main drawback of Drop Tail. Several AQMs come to rescue at this point. The phenomenon once again appears in TCP FreeBSD, the popular algorithm in Unix OS that does not own a saw-tooth graph. Figure 5(b), which is simulated under TCP FreeBSD, also presents a long delay (≈2.8 sec) when buffer is too large. Those results confirm bufferbloat happening at different OS whenever a long buffer queue is built up.

5.3. CPC Performance under TCP Real-World Stacks. Next, we conduct more necessary simulations to compare our CPC scheme performance with traditional popular queuing disciplines Drop Tail and RED. We monitor the queue size at the bottleneck point (router 1), the round-trip time of packets, and goodput at each destination server. Goodput is the application level throughput, that is, the number of user information bits delivered by the network to a certain destination per unit of time. For example, if a file is transferred, the goodput that the user experiences corresponds to the file size in bits divided by the file transfer time. The goodput is always lower than the throughput (the gross bit rate that is transferred physically), which generally is lower than the channel capacity or bandwidth. Goodput monitoring is necessary to evaluate the actual good throughput performance inside networks. We also do these comparisons under several different TCP versions which include Reno, Cubic, and FreeBSD. The first one is mainly implemented in Windows and some Linux Kernels. The second one is improved and deployed in newer versions of Linux Kernel until now. The last one is popularly implemented under Unix Kernel. All the following results are derived when we set the largest buffer size at congestion point, 500 (packets). In the following, we summarize the background knowledge of those three TCP versions before going into detailed results of bufferbloat delay reduction and goodput stability.

5.3.1. Under Reno Real-World Stack. For each connection, TCP maintains a congestion window *cwnd*, limiting the total number of packets that may be transmitted at a time. TCP Reno algorithm uniqueness is the fast recovery phase. That means when a packet loss is detected by receiving three duplicate ACKs, Reno will halve *cwnd* and set slow start threshold *ssthresh* equal to the new *cwnd* value, perform a fast retransmit, and enter a phase called fast recovery. It also reduces congestion window to one maximum segment size (MSS) on a timeout event. Congestion-avoidance phase, as



(a) Round-trip time

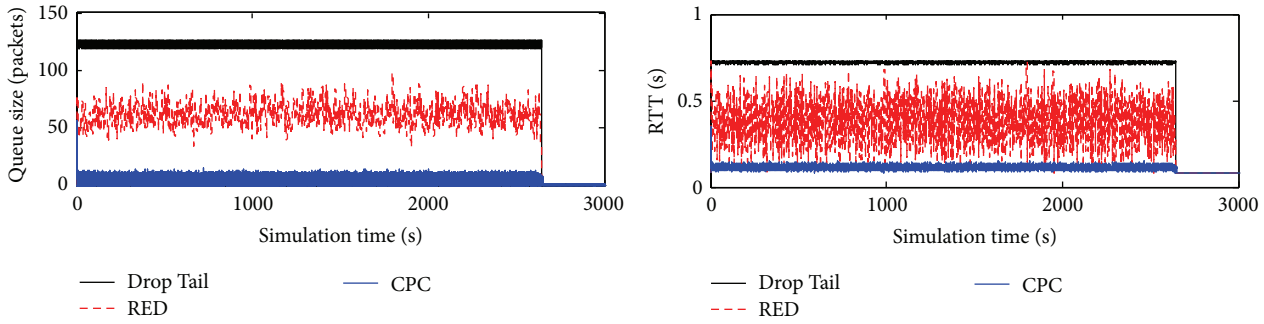


(b) Goodputs

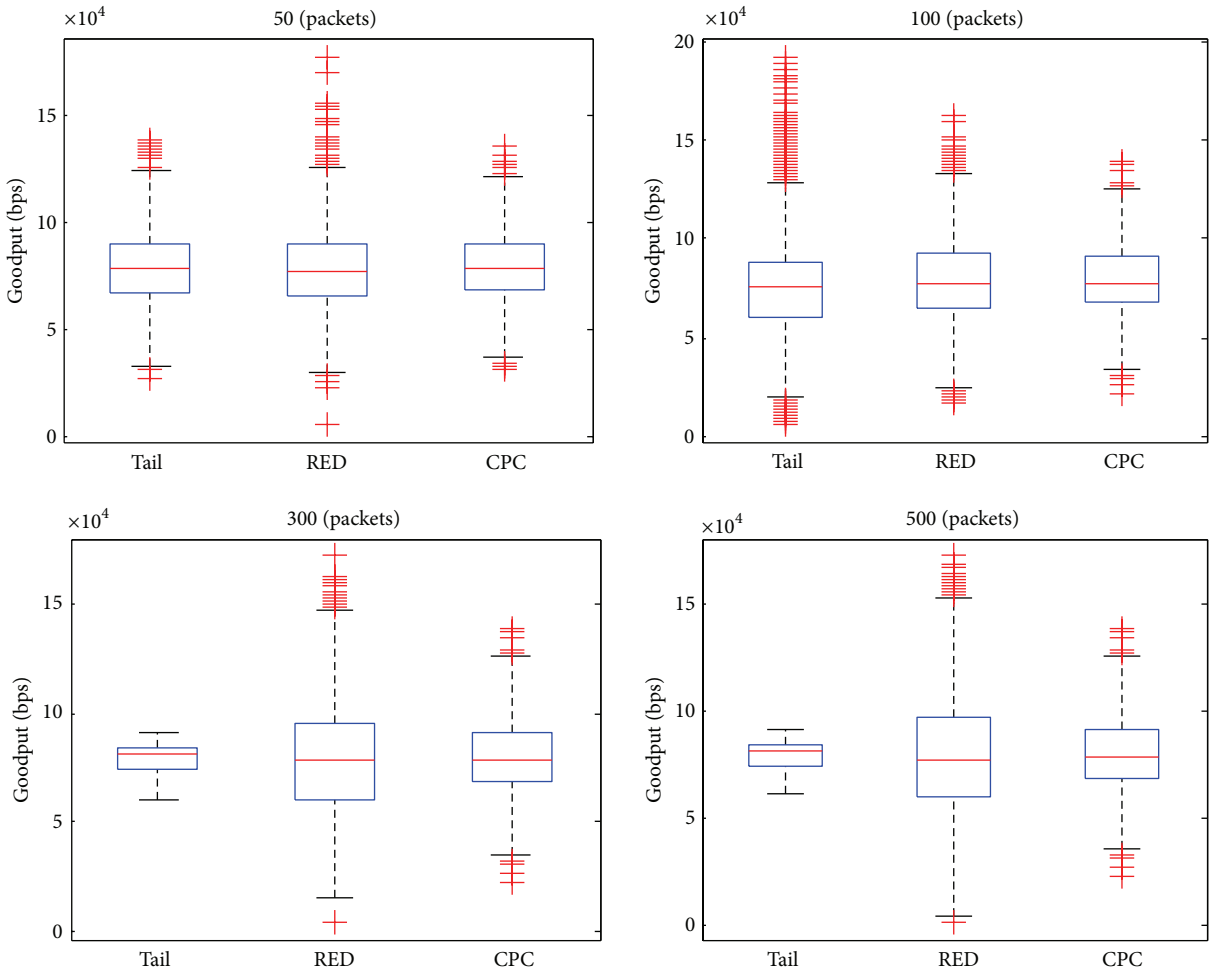
Figure 6: TCP Reno + Drop Tail/RED/CPC.

usual, exploits AIMD to control *cwnd* based on packet loss notification. In fact, compound TCP stack for Windows and Linux also increases AIMD window as TCP Reno does [29].

In Figure 6(a), queue size and RTT according to three queue management schemes are presented. With 500 packets of buffer, Reno and Drop Tail show a saw-tooth graph where the highest RTT is high up to 2.8 (sec). With the target RTT being 200 ms, RED under Reno presents better performance than Drop Tail, RTT (RED) $\approx$ 400 ms, by early dropping some packets to keep the queue size small. CPC under Reno, however, achieves a much better performance while keeping the lowest queue size and lowest RTT (CPC) $\approx$ 180 ms, which is acceptable for delay-sensitive applications. The reason is that CPC continuously controls the next queue size value



(a) Round-trip Time



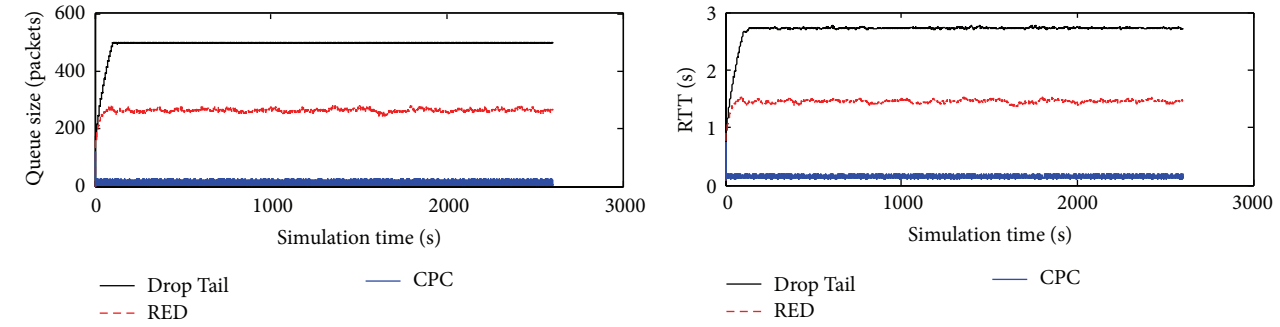
(b) Goodputs

Figure 7: TCP Cubic + Drop Tail/RED/CPC.

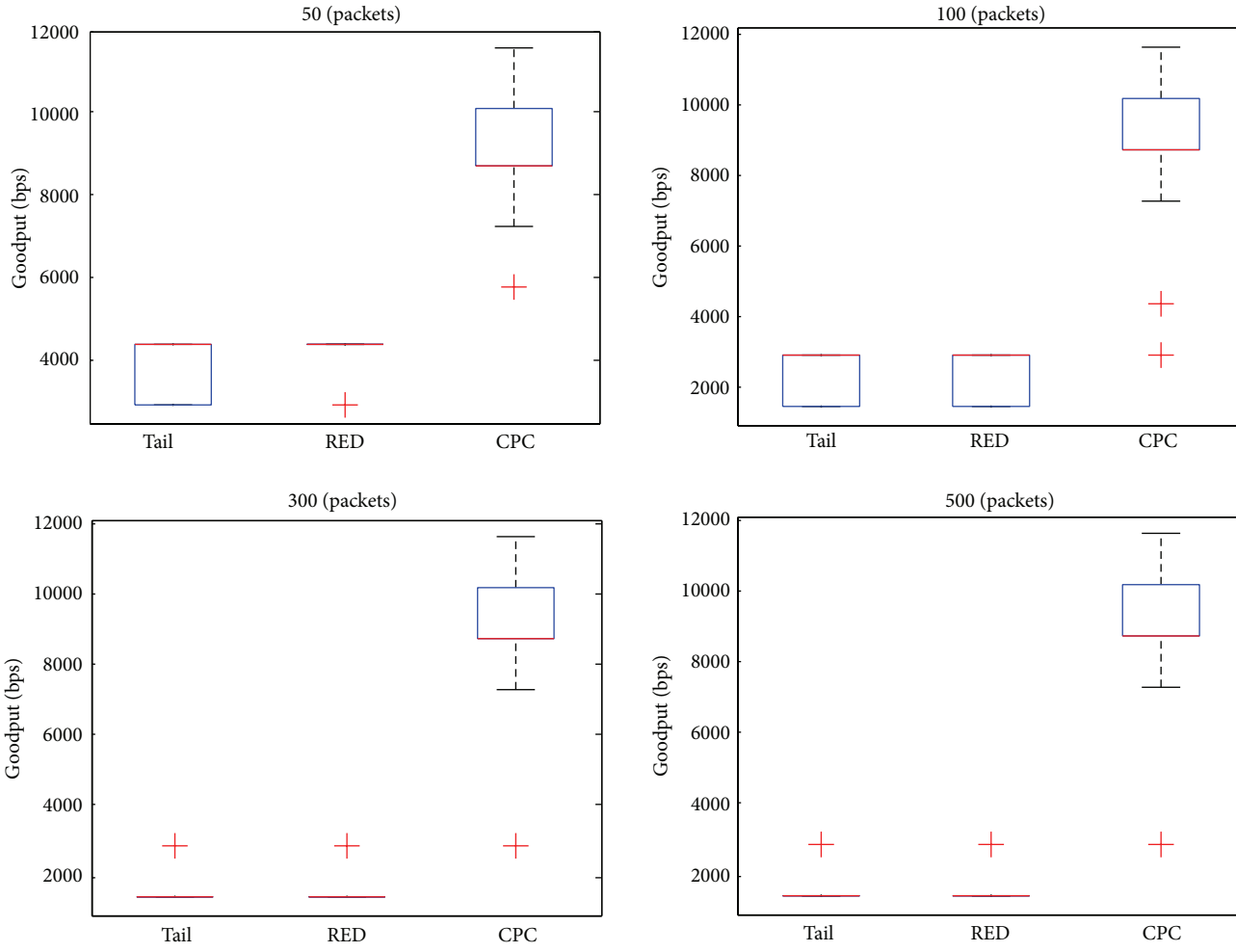
according to traffic changes and adjusts the current queue size at the outer loop so that it indirectly affects RTT value of packets.

Next, Figure 6(b) shows us goodput (bit per second) statistic results under TCP Reno according to four different buffer sizes. These graphs use box-and-whisker plot that can effectively depict groups of numerical data through their quartiles. For instance, the center red line of box represents the median goodput value; the width of box shows the spread degree of goodput statistic data; for example, bigger box and longer whisker mean that statistic data oscillate so much or are unstable. Regardless of different buffer size, under TCP Reno, the median of goodput is nearly the same, around 8×10^4 (bps). The CPC scheme, however, achieves the best stability performance compared to the others because the width of CPC boxes is smallest.

5.3.2. Under Cubic Real-World Stack. Cubic is a congestion control protocol for TCP and the current default TCP algorithm in the Linux kernel [30]. It replaces the default



(a) Round-trip Time



(b) Goodputs

Figure 8: TCP FreeBSD + Drop Tail/RED/CPC.

TCP congestion control algorithm described above with a different algorithm based on a cubic function of the time since the last congestion event. Instead of adjusting the congestion window as a function of previous values as each packet is acknowledged, the Cubic algorithm recomputes the congestion window size at each step using the cubic function calibrated. This results in the congestion window responding quickly to changes in available bandwidth. The TCP Cubic is a less aggressive and more systematic derivative of binary increase congestion (BIC). We exploit NSC module to incorporate real-world TCP/IP network stacks into our simulation model. NSC owns TCP Cubic (version 2.6.26) stack which allows us to test our CPC queuing scheme performance.

In Figure 7(a), we can see again that CPC scheme seems to achieve the best performance, keeping lower queue size and RTT only 130 ms, which is acceptable for delay-sensitive traffic. However, the RTT gaps between three schemes are not

so much because of the aggression property of TCP Cubic. While RED shows large variability of RTT, the others keep RTT much more stable.

Goodput performance in Cubic also reveals TCP Cubic's aggression. When buffers are at small values (50 and 100 packets), three boxes' widths are nearly similar. That means that CPC can achieve the same goodput as the others if we use small buffers. Moreover, increasing the buffer size to 300 and 500 packets makes CPC's goodput more stable, with the smaller box width. These figures demonstrate that CPC can mitigate the bufferbloat—big buffers problem pretty well.

5.3.3. Under FreeBSD Real-World Stack. Finally, we investigate one more popular TCP stack. FreeBSD is a free Unix-like operating system, historically standing for "Berkeley Unix." It was chosen as a BSD-derived TCP/IP stack that is widely used and has had much development. There are several applications that are directly based on FreeBSD, an example being the famous instant messenger WhatsApp. Much of FreeBSD became an integral part of other operating systems such as Apple's OS X. The integrated NSC module for OMNeT++ also incorporates TCP FreeBSD real-world stack so that we can turn it on for demonstrating CPC's performance under FreeBSD. Figure 8(a) presents queue size and RTT results for three disciplines. We can see that our proposal CPC achieves a better performance, keeping lower queue size and RTT (FreeBSD) around 150 ms.

So far, the proposed CPC scheme achieved good goodput performance under TCP Reno and Cubic stacks. We, however, find another interesting fact of CPC when performing under TCP FreeBSD stack. In Figure 8(b), we can conclude two things. Firstly, the width of CPC goodput's boxes in four buffer cases is almost bigger than the other schemes. It means the CPC stability under FreeBSD stack's environment is not good. Secondly, our CPC scheme, however, still achieves more goodput than others. Specifically, in four cases, the system's goodput can reach 10 (kbps) using CPC while it can reach only 4 (kbps) and 2 (kbps) under either small or large buffer using Drop Tail or RED schemes, respectively.

6. Conclusions

Bufferbloat problem, with overfilling a queue at the low-speed side, leads to long end-to-end latency because of the persistent long queuing delay. In this paper, we proposed a control scheme called cascade probability control (CPC) as an alternative way to tackle this issue and reduce queuing delay. Our scheme introduces two-loop control model which consists of queue length and window size control to keep round-trip time value around an expected value. In comparison to current deployed schemes, our scheme's advantages are to maintain stability and noise reduction while the queuing delay can still be ensured under several different TCP versions. Further works would extend CPC on bufferbloat and queuing delay control in the cellular communications.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research was supported by the MSIP (Ministry of Science, ICT, and Future Planning), Korea, under the Global IT Talent support program (R0618-15-1001) supervised by the IITP (Institute for Information and Communications Technology Promotion).

References

[1] J. B. Nagle, "On packet switches with infinite storage," *IEEE Transactions on Communications*, vol. 35, no. 4, pp. 435–438, 1987.

[2] J. Gettys, "Bufferbloat: dark buffers in the Internet," *IEEE Internet Computing*, vol. 15, no. 3, pp. 95–96, 2011.

[3] T. Hoiland-Jorgensen, *Battling bufferbloat: an experimental comparison of four approaches to queue management in linux [Master module project]*, Roskilde University, 2012.

[4] M. Hassan and R. Jain, *High Performance TCP/IP Networking: Concepts, Issues, and Solutions*, Prentice-Hall, 2003.

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

[6] J. Sun, M. Zukerman, and M. Palaniswami, "Stabilizing RED using a fuzzy controller," in *Proceedings of the IEEE International Conference on Communications (ICC '07)*, pp. 266–271, June 2007.

[7] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive red: an algorithm for increasing the robustness of red's active queue management," Tech. Rep., 2001.

[8] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy red," in *Proceedings of the 7th International Workshop on Quality of Service (IWQoS '99)*, pp. 260–262, 1999.

[9] J. Hwang and S.-S. Byun, "A resilient buffer allocation scheme in active queue management: a stochastic cooperative game theoretic approach," *International Journal of Communication Systems*, vol. 28, no. 6, pp. 1080–1099, 2015.

[10] M. Khosroshahy, "UARA in edge routers: an effective approach to user fairness and traffic shaping," *International Journal of Communication Systems*, vol. 25, no. 2, pp. 169–184, 2012.

[11] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, p. 20, 2012.

[12] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "A control theoretic analysis of RED," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '01)*, vol. 3, pp. 1510–1519, IEEE, Anchorage, Alaska, USA, April 2001.

[13] R. Zhu, H. Teng, and J. Fu, "A predictive PID controller for AQM router supporting TCP with ECN," in *Proceedings of the 2004 Joint Conference of the 10th Asia-Pacific Conference on Communications and the 5th International Symposium on Multi-Dimensional Mobile Communications*, vol. 1, pp. 356–360, August 2004.

[14] G. Kahe, A. H. Jahangir, and B. Ebrahimi, "A compensated PID active queue management controller using an improved

queue dynamic model," *International Journal of Communication Systems*, vol. 27, no. 12, pp. 4543–4563, 2014.

[15] H.-L. To, G.-O. Yoon, J.-H. Nam, G. Solongo, and W.-J. Hwang, "Feedback burst loss ratio control for link performance improvement in optical burst switching networks," *Journal of Korea Multimedia Society*, vol. 16, no. 9, pp. 1067–1076, 2013.

[16] K. B. Kim, "Design of feedback controls supporting TCP based on the state-space approach," *IEEE Transactions on Automatic Control*, vol. 51, no. 7, pp. 1086–1099, 2006.

[17] H.-L. To, S.-H. Lee, and W.-J. Hwang, "A burst loss probability model with impatient customer feature for optical burst switching networks," *International Journal of Communication Systems*, vol. 28, no. 11, pp. 1729–1740, 2015.

[18] R. Adams, "Active queue management: a survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1425–1476, 2013.

[19] J. Aweya, M. Ouellette, and D. Y. Montuno, "DRED: a random early detection algorithm for TCP/IP networks," *International Journal of Communication Systems*, vol. 15, no. 4, pp. 287–307, 2002.

[20] F. Ren, C. Lin, and B. Wei, "Design a robust controller for active queue management in large delay networks," in *Proceedings of the 9th International Symposium on Computers and Communications (ISCC '04)*, vol. 2, pp. 748–754, June 2004.

[21] S. Xiang, B. Xu, S. Wu, and D. Peng, "Gain adaptive smith predictor for congestion control in robust active queue management," in *Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA '06)*, vol. 1, pp. 4489–4493, June 2006.

[22] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, 1989.

[23] G. Patil and G. Raina, "Some guidelines for queue management in high-speed networks," in *Proceedings of the 3rd Asian Himalayas International Conference on Internet (AH-ICI '12)*, pp. 1–6, IEEE, Kathmandu, Nepal, November 2012.

[24] F. Du and J. Sun, "An aqm scheme based on adaptive weight cascaded pid controller," in *Proceedings of the 10th World Congress on Intelligent Control and Automation (WCICA '12)*, pp. 2849–2854, July 2012.

[25] N. Kuhn, P. Natarajan, D. Ros, and N. Khademi, *AQM Characterization Guidelines*, IETF89, 2014.

[26] A. Varga, "Using the OMNeT++ discrete event simulation system in education," *IEEE Transactions on Education*, vol. 42, no. 4, p. 11, 1999.

[27] D. Taht, Realtime Response Under Load (RRUL) Test, draft 07, 2012, https://github.com/dtaht/deBloat/blob/master/spec/rrule.doc.

[28] C. Hornig, "A standard for the transmission of IP datagrams over ethernet networks," Network Working Group RFC894, 1984.

[29] D. J. Leith, L. L. H. Andrew, T. Quetchenbach, R. N. Shorten, and K. Lavi, "Experimental evaluation of delay/loss-based TCP congestion control algorithms," in *Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet '08)*, University of Manchester, Manchester, UK, March 2008.

[30] 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.