Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4180817, 10 pages
http://dx.doi.org/10.1155/2016/4180817

Research Article

Optimal Control for Bufferbloat Queue Management Using Indirect Method with Parametric Optimization

Amr Radwan,[1] Hoang-Linh To,[1] and Won-Joo Hwang[2]

[1]*Department of Information and Communication System, Inje University, Gimhae, Gyeongnam 50834, Republic of Korea*
[2]*Department of Information and Communications Engineering, HSV-TRC, Inje University, Gimhae, Gyeongnam 50834, Republic of Korea*

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

Received 20 May 2016; Accepted 31 July 2016

Academic Editor: Junhu Ruan

Because memory buffers become larger and cheaper, they have been put into network devices to reduce the number of loss packets and improve network performance. However, the consequences of large buffers are long queues at network bottlenecks and throughput saturation, which has been recently noticed in research community as bufferbloat phenomenon. To address such issues, in this article, we design a forward-backward optimal control queue algorithm based on an indirect approach with parametric optimization. The cost function which we want to minimize represents a trade-off between queue length and packet loss rate performance. Through the integration of an indirect approach with parametric optimization, our proposal has advantages of scalability and accuracy compared to direct approaches, while still maintaining good throughput and shorter queue length than several existing queue management algorithms. All numerical analysis, simulation in ns-2, and experiment results are provided to solidify the efficiency of our proposal. In detailed comparisons to other conventional algorithms, the proposed procedure can run much faster than direct collocation methods while maintaining a desired short queue (≈40 packets in simulation and 80 (ms) in experiment test).

1. Introduction

Nowadays, modern computer networks are incredibly complex and we rely on them to transport huge quantities of data across the globe in seconds. Although this works well, there are some foreseen issues that need to be tackled. As bandwidth-heavy applications such as peer-to-peer networks and websites relying on user-generated content have become more prevalent, especially the relatively slow residential broadband links have been used at full capacity, and interruptions in connectivity have become more common [1]. Recent research has shown that the culprit is the buffers built into network equipment [2]. Accordingly, *bufferbloat* term has been used to describe related issues whenever these buffers misbehave to produce unnecessary latency [3].

In order to efficiently manage queues which are generated due to the *bufferbloat* phenomenon, active queue management (AQM) algorithms have been recommended to use in network equipment [4]. Most of the existing approaches to AQM design exploit feedback control theory with the linearized TCP core model that was proposed by Hollot et al. [5]. The well-known AQMs that monitor the average queue size and drops (or marks) packets based on statistical probabilities are Random Early Detection (RED) [6] or Random Early Marking (REM) [7]. If the buffer is empty, all incoming packets are accepted. When the buffer is full, the probability has reached 1 and all incoming packets are dropped. When queue is growing, the probability grows according to a piecewise linear function and RED (or REM) drops (or marks) packets using the updated probability. The main drawback of RED or REM is that it is sensitive to network parameter changes and requires careful tuning of its parameters in order to provide optimal performance in any scenarios. Recently, in [8], a Proportional Integral Enhanced (PIE) controller as a lightweight AQM is proposed, without the need of per-packet extra processing. Such PI-type controllers are known to provide queue control with zero offset (the mean queue length converges to the target value)

but are consequently less stable and slower reacting. There have been several other control-theoretic solutions based on feedback fluid-flow model proposed in [9–12] to improve stable and robust control mechanisms, but none of them tackle the issue of system optimality in terms of minimizing both queue length and packet dropping rate performance and searching for an optimal control trajectory.

To address such issues, a promising design direction is to reformulate the network queuing problem as an *optimal* control queue (OCQ) problem [13], where the main state variable is queue length and the control variable is the actual input rate to form the queue [14] or packet dropping rate. Then one of the approaches to solve OCQ is that we can priorly discretize the governing ordinary differential equations (ODEs) and the integral terms in the cost function or constraint functions and thereby replace the infinite dimensional optimal control problem with a large nonlinear optimization problem (NOP). This is known as the *direct method* in the literature for solving OCQ. This approach is typically easier to use, especially for OCQ with a state equality or inequality constraints. The main difference among direct approaches is how to handle the constraints corresponding to the system dynamics. The three most common direct approaches are direct single shooting, direct multiple shooting, and direct collocation. Direct methods have been used with references in [15–18].

Alternatively, one can first form the optimality conditions, using the calculus of variations and Pontryagin minimum principle, and then solve the resulting boundary value problem. This is known as the *indirect method* for solving OCQ. The references present just a small sample of the work that discusses or applies indirect methods for the solution of optimal control problems [15, 16, 18, 19]. In rare cases, the solution can be obtained in closed form from the optimality conditions, but, in general, approximation methods are used to solve the problem numerically. The optimality conditions of these problems generally take the form of differential algebraic equations (DAEs) with boundary conditions (BCs). The approximate solution to the OCQ can be obtained by using a boundary value problem (BVP) solver. Perhaps the most popular methods are multiple shooting and collocation. More recently, a combination of direct and indirect methods was proposed leading to hybrid methods [16].

In this paper, we firstly derive methods to solve OCQ using both direct and indirect approaches. We show how to apply the direct collocation approach to solve OCQ problem which can be solved in popular optimization solvers such as JModelica [20] and GAlib [21]. Indirect method with forward-backward for optimal control queue algorithm (FB-OCQ) is designed as an alternative to tackle it as well. Our key difference from existing works is that we provide a novel method to update the control step δu by solving a parametric optimization subproblem. This method is scalable which means that we can expand for larger problems with more variables as well. The numerical results for both direct and indirect approaches are discussed in Section 5.1 to emphasize our choice of integrating an indirect method with parametric optimization for active queue management design, which is demonstrated more efficiently (i.e., faster reaction and more stable) than direct methods. Finally, we evaluate the proposed algorithm using network simulator ns-2 and compare it to other AQMs including RED, REM, and PI. The dropping feature of the proposed algorithm makes the average queue length shortened and stabilized compared with others, while throughput is not reduced so much.

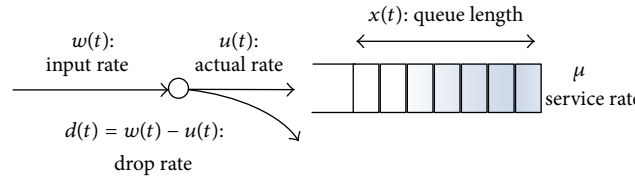

Figure 1: OCQ model.

Our Contributions

(i) We present both indirect and direct methods for an optimal control problem which is applied in queue management field (Section 3). The step update δu of control variable is calculated by solving a subproblem of parametric optimization with the advantage of scalability. Numerical results show that all of them bring nearly similar results, but indirect method (FB-OCQ) is much faster than other solvers and gives the best cost function value (Section 5.1).

(ii) We evaluate FB-OCQ in simulation and show that a desired small queue length value at 40 packets can be obtained. Nevertheless, we cannot avoid the trade-off between queue length and throughput. FB-OCQ's throughput is slightly smaller than RED's one (0.075 versus 0.082 Mbps); however, it can be an acceptable value when compared to REM's and PI's (Section 5.2).

(iii) We implement FB-OCQ in Linux kernel (Ubuntu 16.04) and test it in the worst case using Realtime Response Under Load (RRUL) test suite. The experiment result shows that our algorithm brings low latency ping value compared with the other existing algorithms in Linux kernel (DropTail and RED).

2. Problem Formulation

We consider an optimal control model of queue management problem, named (OCQ), in [14] (Figure 1). The main idea is to minimize the cost function which implies a trade-off between queue length and dropping rate, subject to a dynamic constraint of queue length along time t_0 to t_f as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & J = \int_{t=0}^{t_f} L\left(x\left(t\right), u\left(t\right), t\right) \\ \text{subject to} \quad & \dot{x} = f\left(x\left(t\right), u\left(t\right), t\right), \\ & 0 \le u\left(t\right) \le w\left(t\right), \end{aligned} \tag{OCQ}$$

where $L(x,u,t) = x(t) + R(w - u(t))$ is cost function; $f(x(t),u(t),t) = u(t) - \mu x/(a + x)$; R is weight on dropping rate; t_f is final time; μ is service rate (bandwidth capacity); and a is parameter for different types of queuing model; for example, when $a = 1$, we obtain an M/M/1 queue.

3. Numerical Methods for OCQ

3.1. Direct Method: Collocation Method. In this section, we present how to apply the direct collocation [18] for OCQ. In this method, we discretize the time interval $t_0 = t_1 < t_2 < t_3 < \cdots < t_k = t_f$ into N elements. The state and the total number of packets and control variables at each node are $x_j = x(t_j)$ and $w_j = w(t_j)$ and $u_j = u(t_j)$, such that the state, control, and packets variables at the nodes are defined as nonlinear programming (NLP) variables:

$$Y = [x(t_1), \ldots, x(t_k), u(t_1), \ldots, u(t_k), w(t_1), \ldots, w(t_k)]. \quad (1)$$

The controls are chosen as piecewise linear interpolating functions between $u(t_j)$ and $u(t_{j+1})$ for $t_j \le t \le t_{j+1}$ as follows:

$$u_{\text{app}}(t) = u\left(t_j\right) + \frac{t - t_j}{h_j}\left[u\left(t_{j+1}\right) - u\left(t_j\right)\right], \quad h_j = t_{j+1} - t_j. \quad (2)$$

The value of the control variables at the center $t_{j+1/2}$ is given by

$$u\left(t_{j+1/2}\right) = \frac{u\left(t_j\right) + u\left(t_{j+1}\right)}{2}, \quad j = 1, \ldots, k-1. \quad (3)$$

The piecewise linear interpolation is used to prepare for the possibility of discontinuous solutions in control. Similarly, we can derive the approximate of the total number of the packets $w_{\text{app}}(t)$ and $w(t_{j+1/2})$ as above. The state variable $x(t)$ is approximated by a continuously differentiable and piecewise Hermite-Simpson cubic polynomial between $x(t_j)$ and $x(t_{j+1})$ on the interval $t_j \le t \le t_{j+1}$ of length h_j:

$$\begin{aligned} x_{\text{app}}(t) &= \sum_{r=0}^{3} c_r^j \frac{t - t_j}{h_j}, \\ c_0^j &= x\left(t_j\right), \\ c_1^j &= h_j f_j, \\ c_2^j &= -3x\left(t_j\right) - 2h_j f_j + 3x\left(t_{j+1}\right) - \left(h_j\right) f_{j+1}, \\ c_3^j &= 2x\left(t_j\right) + h_j f_j - 2x\left(t_{j+1}\right) + \left(h_j\right) f_{j+1}, \end{aligned} \quad (4)$$

where

$$f_j = f\left(x\left(t_j\right), u\left(t_j\right), w\left(t_j\right), t_j\right), \quad t_j \le t \le t_{j+1}, \ j = 1, \ldots, k-1. \quad (5)$$

The value of the state variables at the center point $t_{j+1/2}$ of the cubic approximation is

$$x\left(t_{j+1/2}\right) = \frac{x\left(t_j\right) + x\left(t_{j+1}\right)}{2} + \frac{t_{j+1} + t_j}{8}\left(f_j + f_{j+1}\right), \quad j = 1, \ldots, k-1, \quad (6)$$

and the derivative is

$$\dot{x}\left(t_{j+1/2}\right) = \frac{3\left(x_j + x_{j+1}\right)}{2\left(t_{j+1} + t_j\right)} - \frac{1}{4}\left(f_j + f_{j+1}\right), \quad j = 1, \ldots, k-1. \quad (7)$$

In addition, the chosen interpolating polynomial for the state and control variables must satisfy the midpoint conditions for the differential equations as follows:

$$f\left(x_{\text{app}}\left(t_{j+1/2}\right), u_{\text{app}}\left(t_{j+1/2}\right), w_{\text{app}}\left(t_{j+1/2}\right), t_{j+1/2}\right) - \dot{x}_{\text{app}}\left(t_{j+1/2}\right) = 0. \quad (8)$$

Equations (OCQ) can now be defined as a discretized problem as follows:

$$\min \quad f(Y) \quad (9)$$

$$\begin{aligned} \text{subject to} \quad & f\left(x_{\text{app}}(t), u_{\text{app}}(t), w_{\text{app}}(t), t\right) - \dot{x}_{\text{app}} = 0, \\ & x_{\text{app}}\left(t_1\right) - x_1 = 0, \\ & 0 \le u_{\text{app}}(t) \le w_{\text{app}}(t), \end{aligned} \quad (10)$$

where x_{app}, u_{app}, and w_{app} are the approximations of the state, the control, and the total number of packets, constituting Y in (9). This above discretization problem (9)-(10) can be solved using the following:

(i) JModelica, which is a package for simulation and optimization of Modelica models (for more details see [20]);

(ii) GAlib, which is C++ library of genetic algorithm (for more details see [21]).

3.2. Indirect Method: Forward-Backward Sweeping. We solve OCQ by using *indirect method* approach:

(i) forming optimality conditions;

(ii) solving BVP by First-Order Sweeping algorithm.

Let $\overline{x}(t)$ be the adjoint variable. At time t, let $u^*(t)$ denote the optimum controls and let $x^*(t)$ and $\overline{x}^*(t)$ denote the state and adjoint evaluated at the optimum. Using Pontryagin minimum principle we get the following equations.

Hamiltonian Function

$$H(x(t), u(t), \overline{x}(t)) = x(t) + R(w(t) - u(t)) + \overline{x}(t)\left(u(t) - \frac{\mu x(t)}{a + x(t)}\right). \quad (11)$$

Adjoint Equations

$$\dot{\overline{x}}(t) = -1 + u(t)\frac{a\mu}{(a + x(t))^2}. \quad (12)$$

Transversality Condition

$$\overline{x}\left(t_f\right) = 0. \tag{13}$$

Hamiltonian Minimization Condition. Derivative of the Hamiltonian is evaluated to zero at interior points; hence

$$\begin{aligned} &H_u\left(x\left(t\right), u\left(t\right), \overline{x}\left(t\right), \lambda\left(t\right), t\right) \\ &\quad = -R + \overline{x}\left(t\right) - \lambda_1\left(t\right) + \lambda_2\left(t\right) = 0. \end{aligned} \tag{14}$$

If $u^*(t)$ is optimal in (OCQ), then it must satisfy the minimum condition:

$$H\left(x^*, u^*, \overline{x}^*, t\right) = \min_{v \in U} \quad H\left(x, v, \overline{x}, t\right) \quad \forall t \in N. \tag{15}$$

Furthermore, we assume the following:

(H1) the Hamiltonian is strictly convex with respect to control variable u;

(H2) $\widetilde{u}(t) = \arg\min_{u \in U} H(x, u, \overline{x})$, $t \in [t_0, t_f]$ is continuous on $[t_0, t_f]$;

(H3) $u \in C^r[t_0, t_f]$.

Note that $u(t)$ is determined in a unique way for each t since U is convex and H is convex. Now we consider the problem of minimizing the Hamiltonian with respect to p:

$$\min_{u \in U} \quad H\left(x, u, \overline{x}\right) \quad \text{for each } t \in \left[t_0, t_f\right]. \tag{16}$$

Usually in the literature, u is found explicitly as a function $u = u(\overline{x}, x, t)$ and, after substituting it into the system, the problem reduces to the boundary value problem.

Theorem 1. *Assume that conditions (H1)–(H3) hold and problem has an optimal solution (u^*, x^*). Then for a given ϵ there exists a finite discretization*

$$t_0 = \tau_0 < \tau_1 < \cdots < \tau_i < \cdots < \tau_N = t_f \tag{17}$$

and an approximate solution $\widetilde{u}(t), t \in [t_0, t_f]$, such that

$$\left\| u^*\left(t_i\right) - \widetilde{u}\left(t_i\right) \right\| < \epsilon, \quad i = 1, 2, \ldots, N. \tag{18}$$

Proof. Let u^* be an optimal solution of problem. Then (p^*, x^*) satisfies the conditions:

$$\begin{aligned} \dot{x}_i^* &= \frac{\partial H\left(x^*, u^*, \overline{x}^*\right)}{\partial u_i}, \\ \dot{\overline{x}}_i^* &= -\frac{\partial H\left(\left(x^*, u^*, \overline{x}^*\right)\right)}{\partial x_i}, \\ &\qquad i = 1, 2, \ldots, N, \end{aligned} \tag{19}$$

where

$$H\left(x, u, \overline{x}\right) = \sum_{i=1}^{n} \overline{x}_i\left(t\right) f_i\left(x, u\right) - f_0\left(x, u\right), \quad t \in \left[t_0, t_f\right] \tag{20}$$

and (u^*, x^*) satisfies the minimum principle:

$$H\left(x^*, u^*, \overline{x}^*\right) = \min_{u \in U} \quad H\left(x^*, u, \overline{x}^*\right), \quad t \in \left[t_0, t_f\right]. \tag{21}$$

□

(a) First-Order Sweeping Method. We linearize the Hamiltonian around a reference solution $u(t)$ and obtain the following equation for variation $\delta u(t)$:

$$0 = H_u{}^{\top} + H_{uu}\delta u\left(t\right). \tag{22}$$

With the strong Legendre-Clebsch condition, one can approximate the above equation as

$$\delta u\left(t\right) = \arg \min_{\delta u \in \{U - u\}} \quad H_{\text{aug}}\left(x, u + \delta u, \overline{x}, t\right), \tag{23}$$

where

$$H_{\text{aug}}\left(x, u + \delta u, \overline{x}, t\right) = H\left(x, u + \delta u, \overline{x}, t\right) + \beta \left\|\delta u\right\|^2 \tag{24}$$

is the augmented Hamiltonian function and $\beta\|\delta u\|^2$ is a penalty term. This is really parametric optimization, with $\beta = 0$ initially; if U is a convex set and H is a convex function with respect to u, then δu is a descent direction. If $u(t) + \delta u(t)$ does not yield a reduction, then we set $\beta = 1$ and then double it repeatedly until the objective is really reduced.

Remark 2. An alternative way to ensure descent is to apply Backtracking Line-Search Procedure.

(b) Parametric Optimization to OCQ. Now we consider problem (23) as one parametric minimization problem. Since $H_{\text{aug}}(\overline{x}^*, x^*, u^*, \delta u, t)$ is twice differentiable in δu and assumptions (H1)–(H3) hold, we can apply Theorem 1 to the problem. Then as a result, it generates a discretization, $t_0 = \tau_0 < \tau_1 < \cdots < \tau_i < \cdots < \tau_N = t_f$, and corresponding points $\widetilde{u}_i = \widetilde{u}(t_i)$ such that

$$\left\| \delta u^*\left(t_i\right) - \delta\widetilde{u}\left(t_i\right) \right\| < \epsilon, \quad i = 1, 2, \ldots, N, \tag{25}$$

which proves the assertion.

Remark 3. Parametric optimization also can be applied in finding nominal optimal control given in [22]. It is easy to see that, at each iteration k, the Hamiltonian function is a scalar function of $u \in U \subset R^r$ and $t \in T = [t_0, t_f]$; that is,

$$G_k\left(\delta u, t\right) = H_{\text{aug}}\left(\overline{x}^k\left(t\right), x^k\left(t\right), \delta u, t\right). \tag{26}$$

The latter states that $\delta\widehat{u}^k(t)$ must be a minimizer of the following problem:

$$\min_{\delta u \in U} \quad G_k\left(\delta u, t\right), \quad t \in T \tag{27}$$

which is a problem of parametric optimization as formulated in various papers from [23], where the independent variable t is now considered as unknown parameter $t \in T = [t_0, t_f]$. We can also consider a case when the set of admissible controls is time-varying; that is, $U = U(t)$, $t \in T = [t_0, t_f]$. In this case, a general theory of parametric optimization is also applicable for finding the nominal optimal controls.

Let $(x^*, \delta u^*)$ be an optimal process in problem. Introduce the function $f : U \times R \to R$:

$$f(\delta u, t) = -H(x^*, \delta u, \overline{x}^*, t), \quad t \in [t_0, t_f], \tag{28}$$

where a parametric optimization problem is defined as

$$\begin{aligned} &\min_{\delta u \in U} \quad f(\delta u, t), \quad t \in [t_0, t_f], \\ &U = \{\delta u \in R^r : g_i(\delta u) \le 0,\ i \in J\}, \\ &J = \{1, 2, \ldots, s\}. \end{aligned} \tag{29}$$

The KKT conditions for the problem state that

$$\begin{aligned} &D_{\delta u} f(\delta u, t) + \sum_{j \in J} \mu_i D_\delta u g_i(\delta u, t) = 0, \\ &g_i(\delta u, t)\, \mu_i \le 0, \quad \mu_i \ge 0,\ i \in J, \\ &\mu_i g_i(\delta u, t) = 0, \\ &i \in J,\ t \in [t_0, t_f], \end{aligned} \tag{30}$$

where

$$D_{\delta u} f(\delta u, t) = \left(\frac{\partial f(\delta u, t)}{\partial \delta u_1}, \ldots, \frac{\partial f(\delta u, t)}{\partial \delta u_r} \right). \tag{31}$$

Consider the auxiliary parametric optimization problem:

$$\begin{aligned} &\min \quad f(\delta u, t), \quad t \in [t_0, t_f] \\ &\text{subject to} \quad g_i(\delta u) = 0, \quad i \in \widetilde{J} \subset J. \end{aligned} \tag{32}$$

Let $v^0 = (\delta u^0(t), \mu^0(t))$ satisfy the KKT conditions for problem with $\widetilde{J} = J_0$. This system can be written in the following compact notation:

$$F(\delta u, t) = 0, \quad t \in [t_0, t_f], \tag{33}$$

where $v = (\delta u^0(t), \mu^0(t))$. In order to apply Newton's method to system, we have to solve a linear system with $D_{\delta u} F(v(t), t)$ as matrix. The same matrix is used to compute $\dot{v}(t)$:

$$D_{\delta u} F(\delta u(t), t)\, \dot{v}(t) = D_{\delta u} f(\delta u(t), t). \tag{34}$$

Therefore, using the Newton method as corrector, we have

$$\begin{aligned} &D_{\delta u} F\left(\delta u_i^{k_i - 1}, t_i\right)\left(\delta u_i^{k_i} - \delta u_i^{k_i - 1}\right) = -F\left(\delta u_i^{k_i - 1}, t_i\right), \\ &D_{\delta u} F\left(\delta u_i^{k_i - 1}, t_i\right)\left(\dot{\delta} u_i^{k_i - 1}\right) = -\Delta_t F\left(\delta u_i^{k_i - 1}, t_i\right). \end{aligned} \tag{35}$$

4. Proposed Algorithm: FB-OCQ

The *first-order indirect* approach motivates us to design FB-OCQ algorithm. Shortly, this algorithm uses gradient and does forward-backward searching for the optimal control solution. Let us choose an initial control trajectory:

$$(u_0, \ldots, u_N), (\delta u_0, \ldots, \delta u_N);$$

$$(u_{+0} = u_0, \ldots, u_{+N} = U_N);$$

$$J_- = \infty, \gamma, \text{Iter} = 1, \Delta_0, \text{TOL}, \text{MaxIter}, \rho \in [0, 1].$$

From Algorithm 1 (FB-OCQ), it is obvious that the algorithm terminates if the norm of the gradient of the Hamiltonian with respect to u, $\gamma = \|H_u\|_2$, during the run time of the program, is smaller than the tolerance and the parameter β must change at the inner iterations if we do not have a descent direction, so we must divide β by parameter $\rho \in (0, 1)$ until we get a reduction in the cost function. Figure 2 explains our algorithm steps: forward sweep (control variable and cost function value) and backward sweep (adjoint variable) in detail.

(1) **Initialization:**
  $u(t), \delta u(t) = 0, u_+(t) = u(t), \forall t \in [t_0, t_f]$;
  $\mathcal{J}_- = \infty, \gamma = 0, \beta, \rho \in (0, 1)$;
  Iter = 1, MaxIter, TOL.
(2) **while** Iter ≤ MaxIter **do**
(3) **procedure** Forward Sweep
(4) $x_+(t_0) = x_0, \mathcal{J}_+(t_0) = 0$.
(5) **Integrate Forward** $t : t_0 \to t_f$
(6) **if** Iter > 1 **then**
(7) $\delta u(t) = \arg\min_{\delta u} H_{\text{aug}}(x(t), u(t) + \delta u, \overline{x}(t), t)$;
(8) $u_+(t) = u(t) + \delta u(t)$.
(9) **end if**
(10) $\dot{x}_+(t) = f(x_+(t), u_+(t), t)$.
(11) $\dot{\mathcal{J}}_+(t) = l(x_+(t), u_+(t), t)$.
(12) **end procedure**
(13) $\mathcal{J}_+ = \mathcal{J}_+(t_f)$.
(14) **if** $\mathcal{J}_+ < \mathcal{J}_-$ **then**
(15) $\mathcal{J}_- = \mathcal{J}_+, \beta = \beta\rho$.
(16) **else**
(17) $\beta = \beta/\rho$; go to **Forward Sweep**
(18) **end if**
(19) **procedure** Adjoint Sweep
(20) $\overline{x}(t_f) = \phi_x^\top(x_+(t_f)), \gamma(t_0) = 0$.
(21) **Integrate Backward** $t : t_f \to t_0$.
(22) $x(t) = x_+(t), u(t) = u_+(t)$.
(23) $\dot{\overline{x}}(t)^\top = -(\partial H(x(t), u(t), \overline{x}(t), t)/\partial x)$
(24) $\overline{u}(t) = \partial H(x(t), u(t), \overline{x}(t), t)/\partial u$.
(25) $\dot{\gamma}(t) = \|\overline{u}(t)\|^2$.
(26) $\gamma = \gamma(t_f)$.
(27) **end procedure**
(28) **if** $\|\gamma\|$ < TOL **then**
(29) stop;
(30) **end if**
(31) Iter ← Iter + 1.
(32) **end while**

Algorithm 1: FB-OCQ.

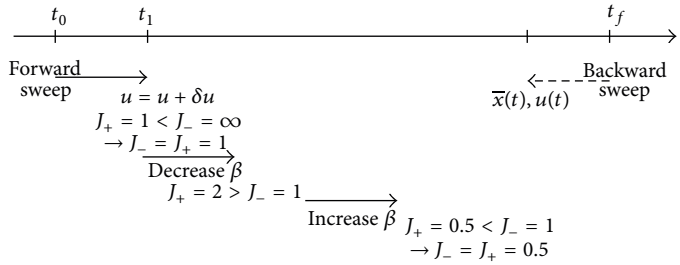

Figure 2: FB-OCQ algorithm explanation.

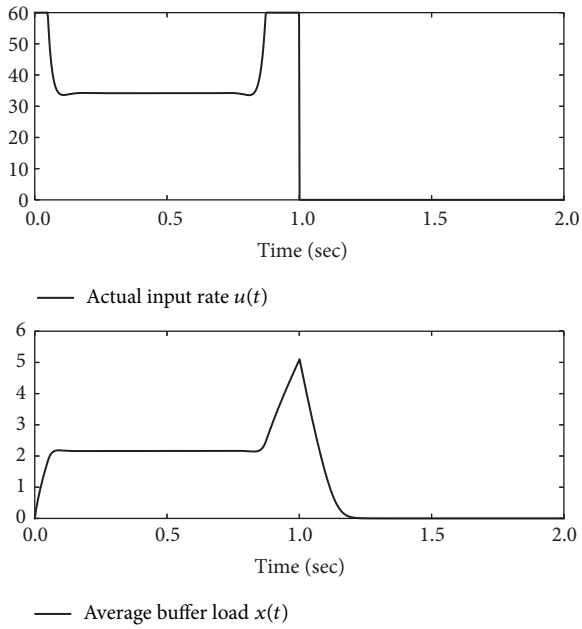

Figure 3: Indirect method: FB-OCQ.

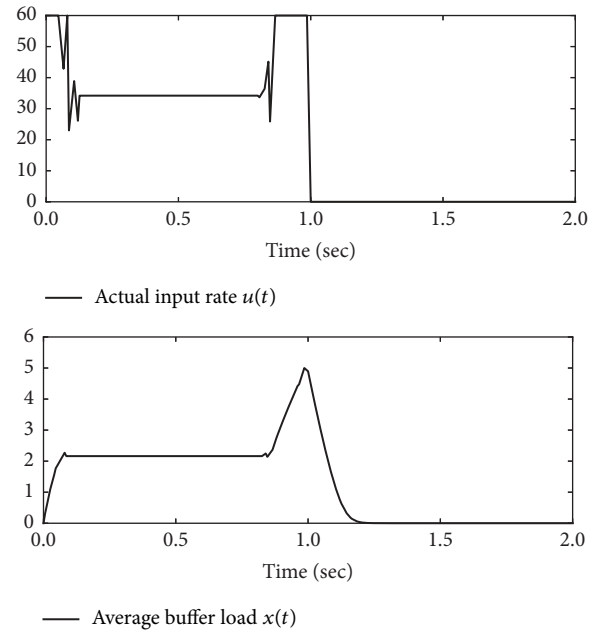

Figure 4: Direct method: nonlinear optimization with JModelica solver.

5. Performance Evaluation

5.1. Numerical Results. In this section, we provide some iteration results for both direct and indirect approaches as follows.

Indirect Method: FB-OCQ. The ODE was solved on equidistant discretization with 1000 discretization points. The optimal control and optimal state are depicted in Figure 3. We realize, by looking at the final control, that there exist some points where the set of active constraints changed due to the singularity; that is, the optimal control is of the bang-bang type with the possibility of a singular arc. To the meaning of such singular arc for queue management, it presents the sudden changes of input rate from 60 (packets/sec) to 35 (packets/sec) which accordingly result in the changes of buffer load (or queue length). In the β history, in Figure 3, the parameter β changed at the inner iterations because we do not have a descent direction, so we must divide β by $\rho \in (0, 1)$ until we get a reduction in the cost function $\mathcal{J} = 6.81280316$. The processing time of central processing unit (CPU) for this algorithm is 0.04 s and the norm of the gradient of the Hamiltonian function $\|H_u\|_2$ with respect to the control u goes to approximately $10e - 7$.

Direct Collocation Method: JModelica and GAlib Solvers. With $N = 1000$ and the number of interpolation points being 3, we tested the discretized problem (9)-(10) by JModelica and GAlib solvers (see [20, 21]) as follows.

(i) JModelica Solver. The cost function is reduced to $\mathcal{J} = 6.8678362$ and the CPU processing time is 0.52 s. The optimal control and optimal state trajectories are obtained during JModelica running and illustrated in Figure 4. An advantage of this approach is that we do not need to derive the adjoint equations.

(ii) GAlib Solver. For this problem, we use the number of generations $n_{\text{gen}} = 2500$, and the population size is 200. The optimal control and optimal state are depicted in Figure 5 obtained during the run of the GAlib. The cost function is reduced to $\mathcal{J} = 7.06744$ during the run time of the program. The CPU processing time is 1.290 s. Although the genetic

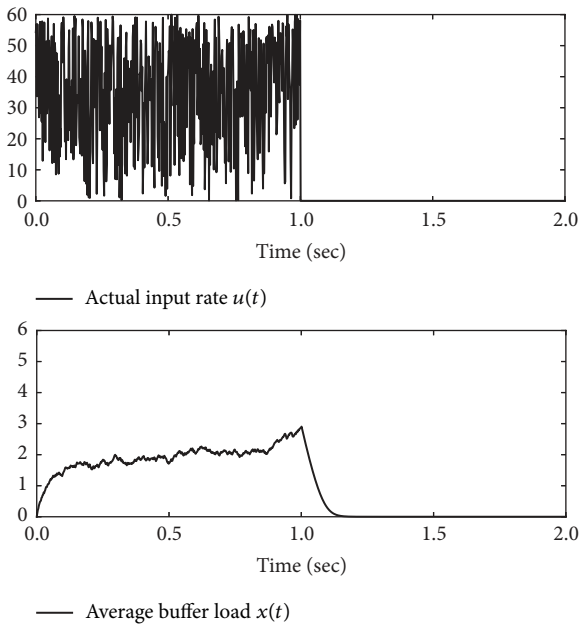

Figure 5: Direct method: genetic algorithm with GAlib solver.

Table 1: Numerical result summary.

Criteria/methods	Direct GAlib	Direct JModelica	Indirect FB-OCQ
Cost function value	$\mathcal{J} = 7.067$	$\mathcal{J} = 6.867$	$\mathcal{J} = 6.812$
CPU process time	1.29 (sec)	0.52 (sec)	0.04 (sec)

algorithm has an advantage that it needs no derivatives or Hessian's information, the control functions produced by it are useless and the convergence is very slow in comparison to Algorithm 1 (FB-OCQ) and the one using JModelica solver.

Table 1 summarizes and compares the three methods that we develop numerically. We conclude that Algorithm 1 is much faster than the other solvers and gives the best cost function. By using genetic library GAlib, the cost function does not reach the local solution as it claims in finding the global solution. The limitation of the indirect methods (FB-OCQ) is that one should derive the adjoint equations which are not easy to derive in some applications.

5.2. Simulation Results. In this section, the performance of the obtained algorithm (FB-OCQ) is evaluated by comparing it with some popular AQMs including RED, REM, and PI. The credibility of results is confirmed using ns-2 simulator [24]. We investigate a network topology with 40 sources, an intermediate router, and one destination (Figure 6). All sources simultaneously send packets to the destination through the router. Hence, a large queue is built up at the router or bottleneck point. Maximum buffer size of the router is 100 packets. All of the compared RED, REM, and PI algorithms are configured to obtain the desired queue length at 40 packets. Simulation lasts for 60 seconds and is repeated using a built-in random generator in ns-2 to obtain more credible results.

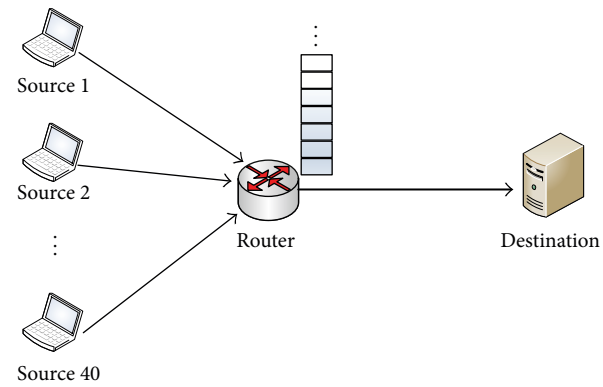

Figure 6: Simulation topology.

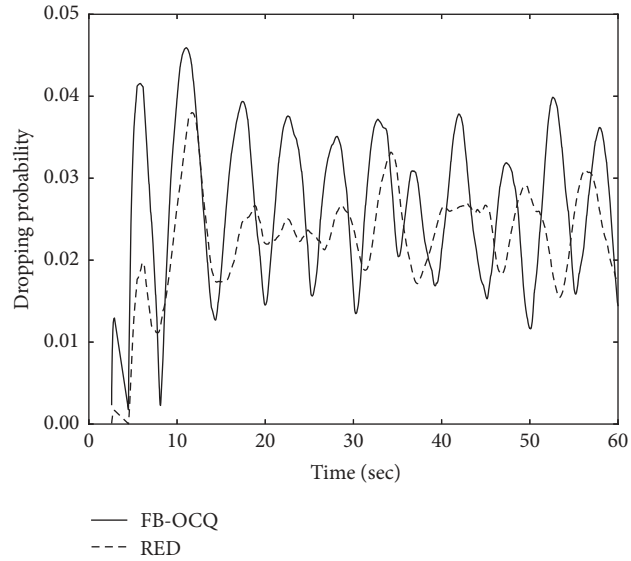

Figure 7: Dropping probability.

Figure 7 shows dropping probability of FB-OCQ and RED algorithms. With the proposed square-root drop function, FB-OCQ drops queuing packets more aggressively than RED although dropping frequency is nearly the same. Maximum dropping ratio is 0.045 for FB-OCQ and 0.039 for RED. In fact, when the algorithm drops more packets, the queue stability will be increased but we will have to sacrifice the system throughput performance. We can see this trade-off in next results.

Figure 8 presents average queue length values measured at the bottleneck link from router to destination for different algorithms. Due to aggressive dropping, FB-OCQ maintains the shortest queue at 40 packets which is also the desired value. Only REM can obtain the same value but in a longer time, ≈50 seconds. PI, in fact, can achieve the same performance only if its parameters are well configured and be dynamically changed to different network scenarios. In Figure 8, we can see that PI performs not so well, even we set the desired queue length variable at 40 packets and exploit default parameters in ns-2 for PI controller.

Finally, we investigate throughput performance of proposed FB-OCQ algorithm. It is confirmed from Figure 9 that there exists a trade-off in the relationship between

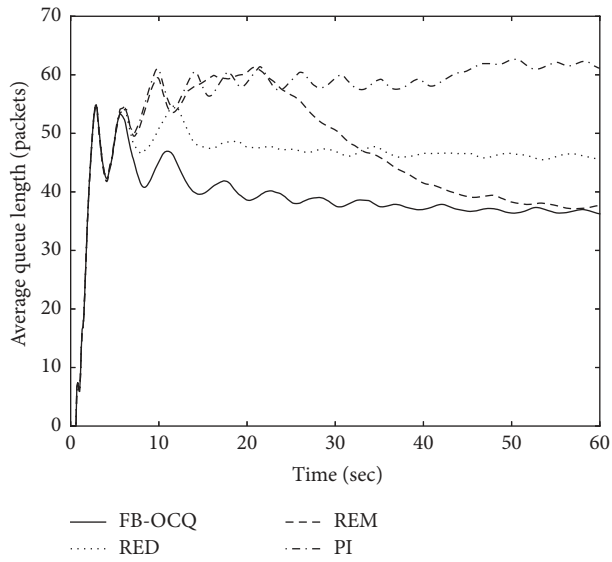

Figure 8: Average queue length versus time.

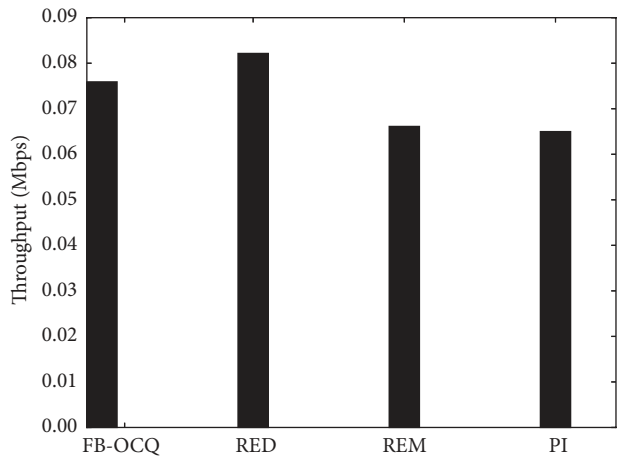

Figure 9: Throughput comparison.

throughput and queue length/dropping probability. PI and REM achieve nearly the same throughput at 0.06 (Mbps) while FB-OCQ and RED obtain the better throughput performance at 0.08 (Mbps) value. In Figure 8, although REM can achieve the desired queue length 40 packets in this scenario, REM still maintains a large queue during simulation time from 0 (sec) up to 25 (sec). That reason leads to throughput performance of REM being lower than RED and FB-OCQ. Our proposed FB-OCQ drops aggressively the packets so that its throughput value (0.075904 Mbps) is slightly less than RED (0.082146 Mbps).

5.3. Experiment Results. We examine the effectiveness of our state-of-the-art optimal control queue method in the Linux kernel (e.g., Ubuntu 16.04). We conduct experiment from a desktop computer through the Internet gateway to an outside server (Figure 10). To manage working queuing disciplines (qdiscs), we use the scheduler qdisc in the Linux kernel. The chosen server is a dedicated bufferbloat server,

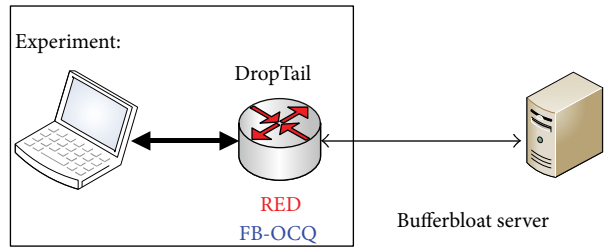

Figure 10: Experiment test setup, from the test client to the test server.

which is able to stand very high congestion due to many data flows at the same time. We exploit the Flent: FLExible Network Tester [25] and Realtime Response Under Load (RRUL) test [26] to evaluate our proposal. RRUL test puts a network under worst case conditions, reliably saturates the network link, and thus recreates bufferbloat phenomenon for queue algorithm testing. Simulation time duration is 60 seconds.

We compare latency under load test with queue being handled by different AQM schemes (DropTail, RED, and FB-OCQ) in turn. DropTail queuing method is by far the simplest approach to network router queue management. The router accepts and forwards all the packets that arrive as long as its buffer space is available for the next incoming packets. If a packet arrives and the queue is currently full, the incoming packet will be dropped. The sender then detects the packet lost event and shrinks its sending window. While it is the most widely used due to simplicity and relatively high efficiency, DropTail has some weakness such as the bad fairness sharing among TCP connections, and throughput and bottleneck link efficiency suffer severe degradation if congestion is getting worse.

RED [5, 6] was presented with the objective of minimizing packet loss and queuing delay. Moreover, it can compensate the weakness of DropTail by avoiding global synchronization of TCP sources so that it improves fairness. To achieve these goals, RED utilizes two thresholds, $\min_{th}$ and $\max_{th}$, and an exponentially weighted moving average (EWMA) formula to estimate average queue length [27]. When the average queue length exceeds a predefined threshold, the link is implied to be in congested state and drop action is taken. A temporary increase in the queue length notifies the transient congestion, while an increase in the average queue length reflects long-lived congestion. Based on such information, RED router sends randomized feedback signals to the senders to make decision of decreasing their congestion windows. RED has good fairness among connections because of the feedback randomized mechanism [28].

Figure 11 presents latency ping results under RRUL test suite. Ping is a networking utility and operates by sending Internet Control Message Protocol (ICMP) echo request packets to the target server and waits for an ICMP echo reply. The program measures the round-trip time from transmission to reception, reporting errors and packet loss. Our proposed algorithm FB-OCQ achieves the lowest packet latency compared with the other two algorithms inside Linux

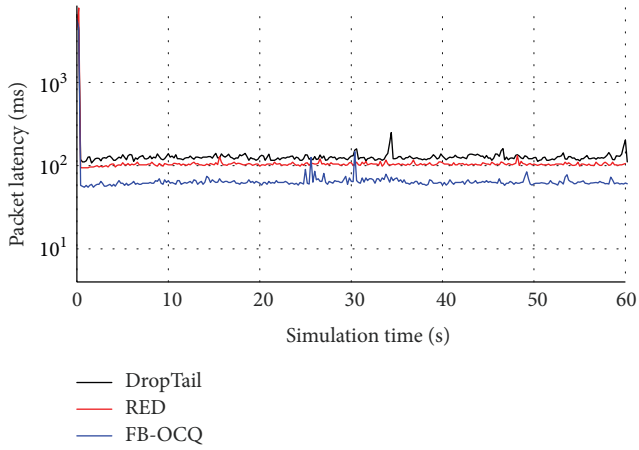

Figure 11: ICMP ping result using Realtime Response Under Load test.

kernel. Specifically, the packet latency when using FB-OCQ is about 80 (ms), while about 120 (ms) if using DropTail (pfifo_fast in Ubuntu) and 100 ms if using RED algorithm.

6. Conclusions

We proposed a queue management algorithm named forward-backward optimal control queue (FB-OCQ) to solve the OCQ problem. Derived from indirect approaches in dynamic optimization, this algorithm demonstrates faster reaction while still achieving the same performance in numerical analysis compared to direct methods. Employing under network simulation ns-2, we see that the proposed algorithm drops packets more aggressively than the traditional RED algorithm, in a higher frequency and magnitude. As a result, average queue length can be reduced much more, while an acceptable value of throughput still can be maintained. In future works, we try to investigate the memory efficiency of FB-OCQ under wireless sensor networks.

Competing Interests

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

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (no. NRF-2016R1A2B1013733).

References

[1] T. Hoiland-Jorgensen, "Battling bufferbloat: an experimental comparison of four apapproach to queue management in linux," Tech. Rep., Roskilde University, Roskilde, Denmark, 2012.

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

[3] J. Gettys, "Bufferbloat: views of the elephant," in *Proceedings of the International Summit for Community Wireless Networks*, October 2012.

[4] B. Braden, D. Clark, J. Crowcroft et al., "Recommendations on queue management and congestion avoidance in the internet," RFC 2309, IETF, San Francisco, Calif, USA, 1998.

[5] C. 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, 2001.

[6] 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.

[7] S. Athuraliya, S. Low, and D. Lapsley, "Random early marking," in *Quality of Future Internet Services*, J. Crowcroft, J. Roberts, and M. I. Smirnov, Eds., vol. 1922 of *Lecture Notes in Computer Science*, pp. 43–54, Springer, Berlin, Germany, 2000.

[8] R. Pan, P. Natarajan, C. Piglione et al., "PIE: a lightweight control scheme to address the bufferbloat problem," in *Proceedings of the IEEE 14th International Conference on High Performance Switching and Routing (HPSR '13)*, pp. 148–155, Taipei, Taiwan, July 2013.

[9] 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-July 2004.

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

[11] 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, Glasgow, Scotland, June 2007.

[12] H.-L. To, T. M. Thi, and W.-J. Hwang, "Cascade probability control to mitigate bufferbloat under multiple real-world TCP stacks," *Mathematical Problems in Engineering*, vol. 2015, Article ID 628583, 13 pages, 2015.

[13] M. Hassan and H. Sirisena, "Optimal control of queues in computer networks," in *Proceedings of the International Conference on Communications (ICC '01)*, vol. 2, pp. 637–641, June 2001.

[14] V. Guffens and G. Bastin, "Optimal adaptive feedback control of a network buffer," in *Proceedings of the American Control Conference (ACC '05)*, vol. 3, pp. 1835–1840, Portland, Ore, USA, June 2005.

[15] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, Society for Industrial and Applied Mathematics, Philadelphia, Pa, USA, 2001.

[16] R. Bulirsch, E. Nerz, H. Pesch, and O. von Stryk, "Combining direct and indirect methods in optimal control: Range maximization of a hang glider," in *Optimal Control*, R. Bulirsch, A. Miele, J. Stoer, and K. Well, Eds., vol. 111 of *ISNM International Series of Numerical Mathematics*, pp. 273–288, Birkhuser, Basel, Switzerland, 1993.

[17] A. Radwan, *Utilization of parametric programming and evolutionary computing in optimal control [Ph.D. dissertation]*, Humboldt-Universität zu Berlin, 2012.

[18] O. von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operations Research*, vol. 37, no. 1, pp. 357–373, 1992.

[19] A. Bryson and Y. Ho, *Applied Optimal Control: Optimization, Estimation and Control*, Hemisphere: CRC Press, Washington, DC, USA, 1975.

[20] M. Ab, Jmodelica.org user guide 1.6.0, 2011, http://www.jmodelica.org/page/236.

[21] M. Wall, *GAlib: A C++ Library of Genetic Algorithm Components*, Version 2.4 Documentation Revision B, Massachusetts Institute of Technology, 1996.

[22] A. Radwan, O. Vasilieva, R. Enkhbat, A. Griewank, and J. Guddat, "Parametric approach to optimal control," *Optimization Letters*, vol. 6, no. 7, pp. 1303–1316, 2012.

[23] J. Guddat, F. Guerra Vazquez, and H. T. Jongen, *Parametric Optimization: Singularities, Pathfollowing and Jumps*, John Wiley & Sons, 1990.

[24] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*, Springer, Berlin, Germany, 1st edition, 2008.

[25] T. Hiland-Jrgensen, Flent: The flexible network tester, 2012–2015, https://flent.org.

[26] D. Tht, "Realtime response under load (rrul) test," 2013, https://github.com/dtaht/deBloat/blob/master/spec/rrule.doc.

[27] J. Koo, S. Ahn, and J. Chung, "A comparative study of queue, delay, and loss characteristics of AQM schemes in QoS-enabled networks," *Computing and Informatics*, vol. 23, no. 4, pp. 317–335, 2004.

[28] T. B. Reddy and A. Ahammed, "Performance comparison of active queue management techniques," *Journal of Computer Science*, vol. 4, no. 12, pp. 1020–1023, 2008.