A Self-Tuning Structure for Adaptation in TCP/AQM Networks

Honggang Zhang, C. V. Hollot, Don Towsley and Vishal Misra

Abstract

Since Active Queue Management (AQM) Schemes are essentially feedback controllers, their impact on closed-loop behavior is directly related to the TCP congestion dynamic which, in turn, is strongly affected by network parameters such as link capacities and TCP loads. These parameters are non-stationary and in the absence of real-time estimates, AQM controllers are designed for worst-case which typically leads to degraded performance under normal situations. Thus, the need for network parameter identification and tuning appears evident, and in this work, we propose a so-called self-tuning AQM (STAQM) that tunes AQM parameters as a function of on-line (and local) estimates of network parameters. This approach is enabled by recent modeling and feedback interpretation of the TCP/AQM dynamic, and, is applicable to any AQM scheme (for example, AVQ, PI, RED and REM) that is parameterizable in terms of link capacity and TCP load. This paper describes the estimation scheme and self-tuning structure and illustrates its use on PI and RED. Analytically, we conduct a local stability analysis of the adaptive system which provides guidelines for choosing the time constant for parameter update. This is crucial, since small time constants could lead to local instability while large ones compromise adaptability. To confirm global performance, we conduct extensive NS simulations where the TCP load varies and experience significant variations in link capacity. Our experiments include heterogeneous round trip times, unresponsive UDP flows, Web traffic, and reverse TCP traffic.

I. Introduction

RECENT advances in the modeling and analysis of TCP/IP networks have led to better understanding of AQM dynamics. Fluid modeling and control theoretic analysis of homogeneous systems [10] has enabled understanding of the relationship between network parameters, such as link capacity C, TCP load N and round-trip time R, to performance objectives, such as AQM responsiveness and robustness. While the analysis dealt with homogeneous systems and long-lived flows, the guiding principles are applicable to realistic network settings with heterogeneous round trip times and short-lived flows. The authors in [10] were able to explicitly tune AQM controllers such as RED, in terms of these parameters, and also conceive of alternative structures, such as PI [9] in a companion paper, for improved performance. The ability of relating AQM design variables to network parameters opens up the ability to tune the designs using realtime estimates of network conditions. This paper is concerned with such adaptive AQM schemes.

The need for AQM adaptability seems evident; mis-tuned AQM controllers significantly degrade performance, and network parameters are quite variable. For example, unresponsive traffic such as short-lived TCP flows, or UDP traffic, effectively alter the link capacity experienced by long-lived TCP flows. Also, traditional analysis has assumed a fixed link capacity while considering AQM design. In practice however, a single physical pipe is often divided into several virtual links using various scheduling mechanisms. The capacities of these virtual links are time varying, depending on the particular scheduling algorithm used. Thus, the assumption of a fixed capacity link is often violated in reality. Likewise, dynamic change in the long-lived TCP workload are to be expected, and the TCP/AQM dynamic is quite

This work is supported in part by DARPA under Contract DOD F30602-00-0554 and by NSF under grants EIA-0080119, ITR-0085848, and ANI-9980552. Any opinions, findings, and conclusions of the authors do not necessarily reflect the views of the National Science Foundation.

Honggang Zhang and Don Towsley are with the Computer Science Department, University of Massachusetts, Amherst, MA 01003; {honggang,towsley}@cs.umass.edu

C. V. Hollot is with the ECE Department, University of Massachusetts, Amherst, MA 01003; hollot@ecs.umass.edu

Vishal Misra is with the Computer Science Department, Columbia University, New York, NY 10027; misra@cs.columbia.edu

sensitive to this variation. A conservative design for the "worst-case" load typically leads to degraded performance under normal scenarios.

This work introduces a self-tuning structure to help AQM controllers deal with variations in link-capacity and TCP workload, which is called STAQM. The scheme has two features: parameter estimation, and AQM tuning. The concept of parameter estimation is to approximate C, N and R from measurements made locally at the congested router. For R, we assume that the queuing delay is dominated by the propagation delay, a reasonable assumption for current and future high capacity links, and then we can bound the two way propagation delays with physical constraints. The actual design of a Self-Tuning AQM should be based on the "effective" RTT [1] since realistic network conditions have heterogenous TCP flows with different RTTs. An estimation scheme based on the router sampling the SYN packets and SYN ACKs of the ongoing TCP flows can be constructed, but that is not the focus of this paper. The link capacity C can be measured by keeping track of departed packets, and, an estimate of the TCP load N/R can be inferred from measurements of the (computed) dropping probability p.[2] Indeed, with (C, p) in hand, the TCP throughput equation $\frac{N}{RC} = \sqrt{\frac{p}{2}}$ provides a means for estimating N/R. This is of particular importance since the quantity N/R in the analysis of [10] is the *aggregate* incoming throughput, or the load on the router. By basing this quantity on an estimate, we are effectively moving beyond the assumption of the analysis in [10] of the mythical N long lived flows with identical round trip times R to the real effective load on a router. Unfortunately, the term R also makes an appearance in the delay term of the dynamical system described in [10] so we still need a separate estimator for R. In the absence of a measured estimate, a worst case estimate for R can be used, but we stress that in the rest of the paper the quantity N/R is to be interpreted as the estimated load on the router, encompassing the effects of both short-lived flows as well as heterogeneous RTTs. The second feature of the self-tuning structure is to automatically tune an AQM based on these estimates. This requires an explicit parameterization of the AQM controller in terms of C, N and R. Such parameterizations are available for both RED and PI.

Related work on adaptive AQM includes Adaptive Virtual Queue (AVQ) [12], Predictive AQM (PAQM) [13], Adaptive RED (ARED) [7] and [2], and self-configuring PI [5]. Our self-tuning structure is most closely related to [5]. It differs in that we incorporate estimates of link capacity, provide stability analysis, and gives guidelines of how to choose filter time constants which is crucial to the stability and responsiveness of STAQM structure. Also, our self-tuning structure can be applied to both PI and RED, which will be discussed later in the paper.

In the paper we will describe, analyze and simulate this self-tuning scheme. On the analysis side we will show how this adaptive controller achieves local exponential stability by choosing appropriate filter time constants for parameter update, and we address the tradeoff between stability and responsiveness. We focus our analysis on the general STAQM structure and its variations on RED and PI, which we call STRED and STPI. We give a comparison of STPI and STRED. In addition, analysis of fixed RED gives a range of network conditions where a stable RED has reasonable responsiveness, and we design STRED in that range. Through `ns-2` simulations, we will compare performance of a STPI and a STRED against a non-adaptive PI controller, and against adaptive RED (ARED). Heterogeneous RTTs are assumed in all the simulations. The scenarios include richer traffic mix such as Web traffic, unresponsive UDP flows, and reverse TCP traffic. The simulation results are encouraging and show that STAQM is robust to large, dynamic variations in both link capacity C and TCP load N.

The paper is organized as follows. In Section II we present some motivating simulations of the self-tuning AQM structure. In Section III, we show how to design a self-tuning AQM for local exponential stability, and in Section IV we put this adaptive scheme through its paces via `ns-2` simulations.

II. Motivation for A Self-Tuning AQM

Before getting into the details of designing a self-tuning AQM, we first give some simulation results to show its benefits over non-adaptive AQM. We will consider situations where link capacity C and TCP flows N vary. We take

[1] [9] suggests a harmonic mean of RTTs as a good candidate for an "effective" RTT.

[2] This is the loss probability computed by the AQM controller, which is readily available.

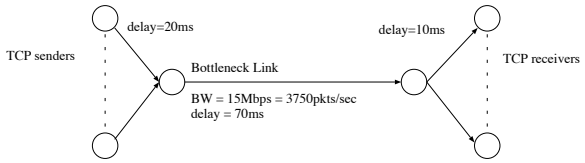

Fig. 1. Simulation settings of s single bottleneck link.

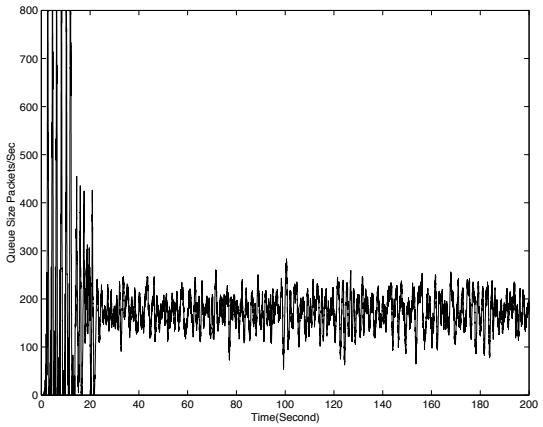

Fig. 2. Instantaneous queue size when using fixed PI designed for N=60, C=15Mbps, and RTT=250ms. N, C, RTT do not change over time.

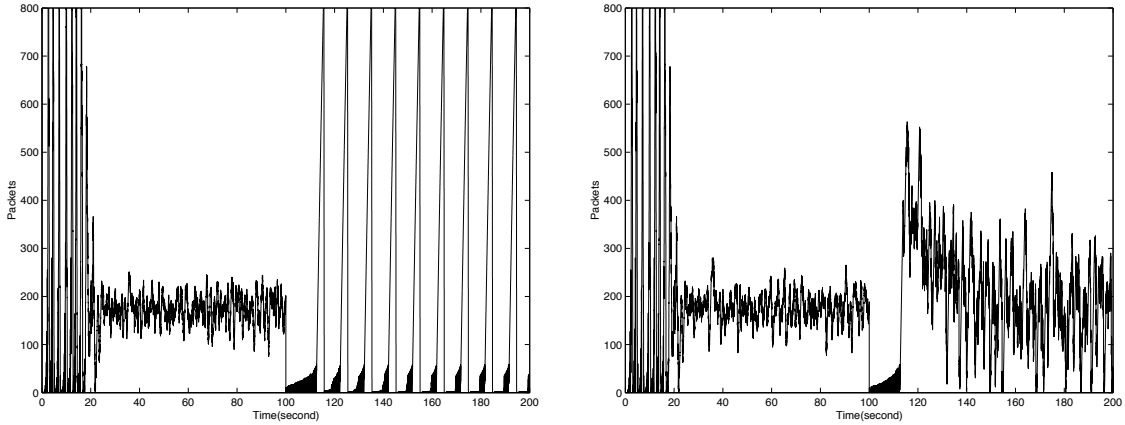

Fig. 3. Comparison of instantaneous queue size between STPI and fixed PI. C increases to 90Mbps from 15Mbps at time 100 seconds. The left plot shows a fixed PI, which is designed for N=60, C=15Mbps, and RTT=250ms. The right plot shows STPI.

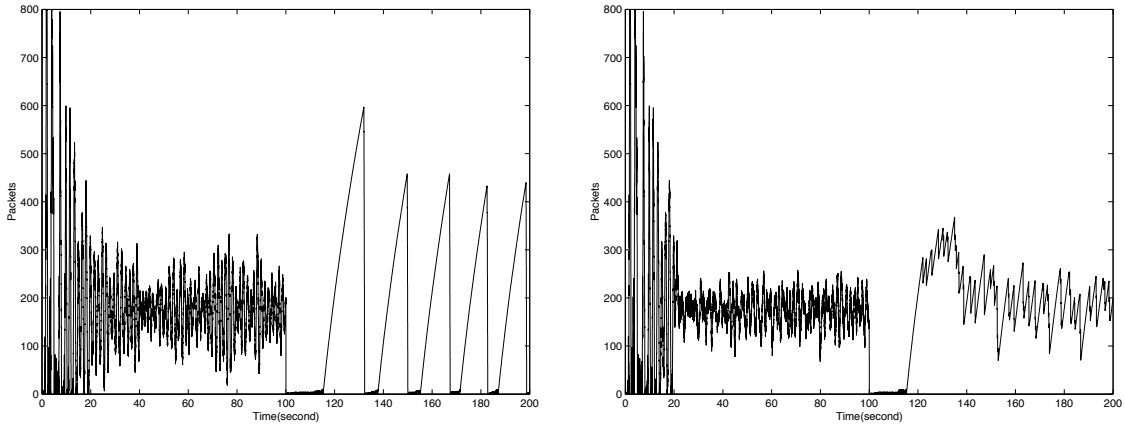

Fig. 4. Comparison of instantaneous queue size between STPI and fixed PI. The number of flows decreases from 240 to 10 at time 100 seconds. The left plot shows a fixed PI, which is designed for N=60, C=15Mbps, and RTT=250ms. The right plot shows STPI.

STPI and fixed PI as examples of self-tuned and fixed AQMs. Figure 1 shows the setup of a single congested link in a dumbbell topology, with two-way propagation delay of 200ms. We assume the fixed PI is designed for a link capacity of 15Mbps and 60 long-lived TCP flows. The target queue size is 175 packets (giving a queuing delay of about 50ms). The router's buffer size is 800 packets and TCP flows randomly start every 20 seconds with packet size fixed at 500 bytes. Following the design rules given in [11], we design PI for these specific values of N, C and R. In `ns`, the PI

coefficients are taken to be $a = 0.000078530$ and $b = 0.000078272$ with a sampling frequency of 160Hz. As shown in Figure 2, PI works well when N and C are at their design values.

Figure 3 shows another simulation where C changes six-fold from 15Mbps to 90Mbps at time 100 seconds. We see that the fixed PI controller loses control of queue length, behaving like a droptail queue. The reason for this oscillation is that the TCP dynamic has a relatively low gain at $C = 15$Mbps, and PI compensates by being aggressive. However, this AQM design turns out to be much too aggressive when gain of the TCP dynamic becomes high when the link capacity increases by a factor of six. Now, we implement STPI as shown in Figure 3, and observe that it regulates the queue in spite of the sudden change in link capacity. The gain of STPI automatically changed to account for the change in C.

Now we fix the link capacity and change the number of TCP flows. We initially take $N = 240$ and accordingly re-tune the fixed PI parameters to $a = 0.00031567$ and $b = 0.00031154$. At time 100 seconds, N decreases to 10 flows. As shown in Figure 4, PI initially stabilizes the queue, but becomes unstable when N suddenly decreases at time=100 seconds. In contrast, STPI successfully adapts to this traffic change at 100 seconds.

Besides PI, we also would like to consider the performance of adaptive RED (ARED) which is an AQM adapting its gain to changes in average queue length. However, it suffers from the fundamental limitations of RED. Here, we would like to show its limitations through simulations. We consider the same topology as in Figure 1, where the two-way propagation delay is now 145ms, queuing delay 5ms, buffer size of 1800 packets, and link capacity C of 300 Mbps. For ARED, we use the guidelines given in [2] and take $min_{th} = 200$ packets, $max_{th} = 600$ packets and $w_q = 0.00001333$. To make the queue length comparable to ARED, we chose STPI's target queue length to be 400 packets. A comparison of performance for STPI and ARED are shown in Figure 5. Initially, 1000 TCP flows start with 900 dropping-out at 100 seconds, and then returning at 200 seconds. As shown in Figure 5, ARED's queue length initially oscillates between 0 and 1000 packets, and then oscillates between 0 and 700pkts when N decreases to 100 flows. In contrast, STPI successfully regulates the queue length to its target queue length of 400 packets, in the face of the varying TCP load.

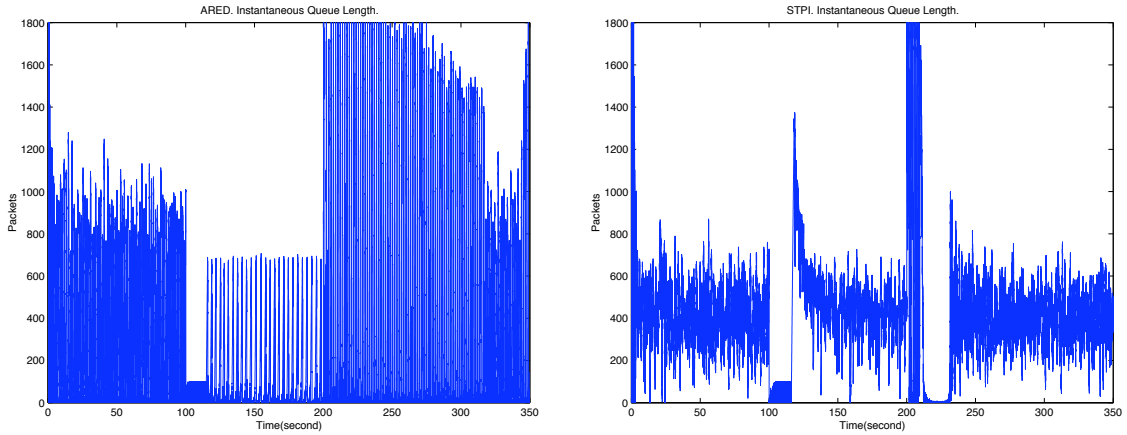

Fig. 5. Comparison of instantaneous queue size between STPI and ARED. Left plot is ARED, and right plot is STPI. The number of flows decreases from 1000 to 100 at time 100 seconds, and it increases to 1000 again at time 200 seconds. ARED is designed for C=300Mbps, and RTT=150ms.

In addition to the simulations presented here, we will also demonstrate the performance of self-tuning AQM via simulations in Section IV. The most important performance metrics we are interested in are the link throughput, queuing delay and its variation. Specifically, we will present the simulations in scenarios where link-capacity changes induced by variations in unresponsive traffic, or traffic load changes, or a richer traffic mix including both reverse traffic and http traffic. In all these simulations, we could use heterogeneous long-lived TCP flows with round trip delays uniformly distributed in (40, 250)ms. The simulations have demonstrated that a self-tuning AQM is able to regulate the queuing delays to the target delay and with much less variations than previously proposed AQM's, and achieve the highest throughput. We now present details behind the design of our self-tuning AQM scheme.

III. Design and Analysis of a Self-Tuning AQM

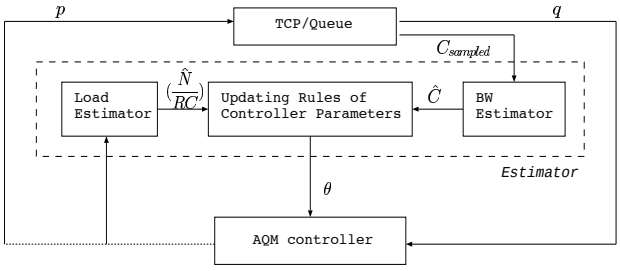

Fig. 6. Block Diagram of a Self-Tuning Controller Model.

In the preceding section we motivated the utility for automatically tuning AQM dynamics to deal with varying network conditions. To formalize the synthesis of such an adaptive scheme we require: (i) a parameterization of AQM dynamics in terms of network variables, (ii) a scheme for on-line parameter identification, and (iii) a rule for self-tuning the AQM dynamics. For this last task, we will lean on the so-called *certainty equivalence principle* found in the classic self-tuning approach to adaptive control [14], and update the AQM dynamic based on the most recently available parameter estimate, proceeding as if this estimate was dead-on. Figure 6 is a control block diagram representation of this self-tuning AQM scheme.

In this section we will first recall how AQM schemes like RED and PI can be parameterized in terms of network variables. Then, we will describe an approach to estimating the link capacity C and TCP load N. Finally, we will show how to select the estimator's time constant to avoid local instability. This is important because incorrectly selected time constant could lead to instability. In the next section we will study the global behavior of this adaptive AQM through `ns` simulations.

A. Parameterization of AQM Dynamics

A generalized fluid description of AQM dynamics relating instantaneous queue length q to loss probability p is

$$\dot{x}_{aqm} = f(x_{aqm}, q); \quad p = g(x_{aqm}, q) \tag{1}$$

where x_{aqm} denotes the AQM state, and f and g describe the AQM dynamic behavior. For example, in RED, x_{aqm} is the average queue length and

$$\dot{x}_{aqm} = -K_{red}\, x_{aqm} + K_{red} q; \quad p = L_{red}(x_{aqm} - q_{ref}) \tag{2}$$

where $K_{red} = -C ln(1 - w_q)$, $L_{red} = \frac{max_p}{max_{th} - min_{th}}$ and $q_{ref} = min_{th}$.[3] In [11], tuning of the RED parameters (K_{red}, L_{red}) is linked to the network parameters (N, R, C) via the relations

$$\omega_g << \min\left\{\frac{2N}{R^2C}, \frac{1}{R}\right\}; \quad K_{red} = \frac{\omega_g}{tan(\beta_{red} - \omega_g R)}; \quad L_{red} = \frac{\sqrt{(\omega_g^2 + K_{red}^2)}(2N)^2}{K_{red}(RC)^3}$$

where $\beta_{red} \in (0, \pi)$ is a designer-selected *responsiveness* factor with larger β_{red} giving more responsive AQM.

For PI AQM, x_{aqm} is the integral of the deviation $(q - q_{ref})$ and

$$\dot{x}_{aqm} = q - q_{ref}; \quad p = K_i x_{aqm} + K_p(q - q_{ref}). \tag{3}$$

The PI parameters (K_p, K_i) are related to network parameters through

$$K_p = 2\beta_{pi}\sqrt{\beta_{pi}^2 + 1}\frac{N}{R^2C^2}; \quad K_i = \frac{2N}{R^2C}K_p$$

[3] In RED, $0 < w_q < 1$ denotes the averaging weight, while min_{th}, max_{th}, and max_p describe the packet-dropping profile.

where $\beta_{pi} \in (0, 0.85)$ controls AQM responsiveness.

The preceding gives examples of how an AQM dynamic can be explicitly related to network parameters (N, R, C). In the absence of knowledge of these network variables, we would like to develop their on-line estimates, and, in the next subsection, we do so for link capacity C and the TCP load N.

B. Estimating Link Capacity and TCP Load

Estimating dynamic variations in link capacity C is both straightforward and accurate. On the other hand, it appears more difficult to track changes in TCP workload. In fact, rather than working with a classical, model-based method for parameter estimation we will simply use the TCP *throughput* relation to back-out an estimate of TCP load. In addition, if the queuing delay is negligible compared with two-way propagation delay, we could use two-way propagation delay to approximate R. As explained in the introduction, we could also use the "effective" RTT or an estimate of RTT made from samples of the SYN packets.

1) Estimating link capacity: To estimate the link's capacity, call it $\hat{C}$, we periodically compute the ratio of departed TCP packets to the router's busy time. Since the transient link capacity of TCP traffic could change due to other rate-varying traffic (such as UDP traffic), we smooth the sampled TCP capacity $\hat{C}$. We use a low pass filter (LPF) to do this and describe it with the fluid model

$$\dot{\theta}_c = -K_c\theta_c + K_c\hat{C}$$

where θ_c denotes the estimated (smoothed) capacity and K_c the filter time constant.

2) Estimating TCP workload: Based on the TCP fluid model in [8] (similar models for the TCP window-control mechanism can be found in [3], [4]), we can derive the following steady-state TCP relationships

$$0 = \frac{1}{R} - \frac{W_0^2}{2R}p_0; \quad 0 = \frac{N}{R}W_0 - C$$

where W_0 is the equilibrium congestion window size (packets) and p_0 is the equilibrium drop probability. From the above, we obtain $\sqrt{p_0/2} = \frac{N}{RC}$. This relationship is a property of TCP, and is *independent* of AQM. Even though the square root relationship is valid only in the steady state, we will use it to estimate $\frac{N}{RC}$ in transient phases. Estimates of C and $\frac{N}{RC}$ are sufficient data to tune RED and PI as previously described. We note that measuring packet loss at the AQM results in an underestimation of actual loss probability which in turn underestimates N. Finally, we smooth the estimate of $\frac{N}{RC}$ using a LPF to give the dynamic:

$$\dot{\theta}^n_{rc} = -K^n_{rc}\theta^n_{rc} + K^n_{rc}\sqrt{p/2}$$

where θ^n_{rc} is the smoothed estimate of $\frac{N}{RC}$ and K^n_{rc} the time constant. This TCP workload estimation scheme first appeared in [5].

C. Combined parameter estimator and tuned AQM

Combining the parameter estimators and AQM model, we have the following model to describe the dynamics of a self-tuning AQM:

$$\begin{aligned} \dot{\theta}_c &= -K_c\theta_c + K_c\hat{C}; \\ \dot{\theta}^n_{rc} &= -K^n_{rc}\theta^n_{rc} + K^n_{rc}\sqrt{p/2}; \\ \dot{x}_{aqm} &= f_\theta(x_{aqm}, q); \\ p &= g_\theta(x_{aqm}, q); \end{aligned} \tag{4}$$

where f_θ and g_θ show the explicit dependence of AQM dynamics on the estimated variables $\theta = (\theta_c, \theta^n_{rc})$. Note that the above AQM equations describe a *nonlinear* dynamic system, even if f and g are linear relationships as in the case of RED and PI. For example, for a PI AQM:

$$\dot{x}_{aqm} = q - q_{ref} \equiv f_\theta; \quad p = K^\theta_i x_{aqm} + K^\theta_p q \equiv g_\theta. \tag{5}$$

where

$$K_p^\theta = 2\beta_{pi}\sqrt{\beta_{pi}^2+1}\frac{\theta_{rc}^n}{R\theta_c}; \quad K_i^\theta = \frac{2\theta_{rc}^n}{R}K_p^\theta.$$

For a RED AQM:

$$\dot{x}_{aqm} = -K_{red}^\theta x_{aqm} + K_{red}^\theta q \equiv f_\theta; \quad p = L_{red}^\theta(x_{aqm} - q_{ref}) \equiv g_\theta. \tag{6}$$

where

$$\omega_g^\theta << \min\left\{\frac{2\theta_{rc}^n}{R}, \frac{1}{R}\right\}; \quad K_{red}^\theta = \frac{\omega_g^\theta}{tan(\beta_{red} - \omega_g^\theta R)}; \quad L_{red}^\theta = \frac{4(\theta_{rc}^n)^2\sqrt{(\frac{\omega_g^\theta}{K_{red}^\theta})^2+1}}{R\theta_c}.$$

Thus, STPI is defined by (4) and (5), while STRED is described by (4) and (6).

D. Selection of Filter Time Constant K_{rc}^n

The objective for introducing adaptation in AQM is to improve performance under network variations. More specifically, an ultimate goal would be to show, analytically, that a self-tuned TCP/AQM network, described by the combined differential equations

$$\begin{array}{ll} \dot{W}=\frac{1}{R} - \frac{W^2}{2R}p_R; & \dot{\theta}_{rc}^n=-K_{rc}^n\theta_{rc}^n + K_{rc}^n\sqrt{p/2}; \\ \dot{q}=\frac{N}{R}W - C; & \dot{x}_{aqm}=f_\theta(x_{aqm}, q); \\ \dot{\theta}_c=-K_c\theta_c + K_c\hat{C}; & p=g_\theta(x_{aqm}, q); \end{array} \tag{7}$$

outperforms a fixed TCP/AQM scheme. Such analysis is beyond the scope of this report and is a topic for future research. In the next section, we will report on the results of `ns` simulations which will provide some experimental justification for employing an adaptive AQM. In this subsection we will provide some guidance in selecting the filter time constants $\frac{1}{K_{rc}^n}$. Our criteria for selecting these constants will be the local stability of (7), and our results will confirm intuition which suggests that, for stable behavior, the time scale for adaptation should be significantly longer than the time scale of AQM responsiveness.

To begin this analysis, we fix parameters (N, R, C) and consider the linearization of (7) around its equilibrium point. Based on the linearizations of STAQM (see Appendix for details), we can draw their control block diagrams in Figure 7 and Figure 8. Now, let $P(s)$ and $C_{aqm}(s)$ be the transfer functions of the TCP and AQM dynamic respectively:

$$P(s) = \frac{\frac{C^2}{2N}}{(s+\frac{2N}{R^2C})(s+\frac{1}{R})}; \quad C_{aqm}^{red}(s) = \frac{K_{red}L_{red}}{s+K_{red}}; \quad C_{aqm}^{PI}(s) = K_{PI}\frac{s/z+1}{s} \tag{8}$$

In (7) we show that

$$\left.\frac{\partial L_{red}}{\partial\theta_{rc}^n}\right|_0 (q_0 - q_{ref}) = \left.\frac{\partial K_i}{\partial\theta_{rc}^n}\right|_0 x_{aqm_0} = 2\sqrt{2p_0}.$$

Consequently, for stability considerations, Figures 7 and 8 reduce to Figure 9.

We now show that Figure 9 is true for a general form of STAQM. Consider a linear state space form of (1)

$$\begin{aligned} \dot{x}_{aqm} &= A(\theta)x_{aqm} + B(\theta)(q - q_{ref}) \\ p &= C(\theta)x_{aqm} + D(\theta)(q - q_{ref}) \end{aligned} \tag{9}$$

Based on the linearization of (7) and (9), we have the following formulas:

$$\dot{\delta W(t)} = -\frac{2N}{R_0^2C}\delta W - \frac{R_0^2C^2}{2N^2}\delta p_R \tag{10}$$

$$\dot{\delta q(t)} = \frac{N}{R_0}\delta W - \frac{1}{R_0}\delta q(t) \tag{11}$$

$$\delta\dot{\theta}_c = -K_C\delta\theta_C \tag{12}$$

$$\delta\dot{\theta}^n_{rc} = -K^n_{rc}\delta\theta^n_{rc} + \frac{K^n_{rc}}{2\sqrt{2}\sqrt{p}}\delta p \tag{13}$$

$$\delta p = \left[\frac{dC}{d\theta}x_{aqm} + \frac{dD}{d\theta}(q - q_{ref})\right]\Big|_0 \delta\theta + C_0\delta x_{aqm} + D_0\delta q \tag{14}$$

$$\delta\dot{x}_{aqm} = \left[\frac{dA}{d\theta}x_{aqm} + \frac{dB}{d\theta}(q - q_{ref})\right]\Big|_0 \delta\theta + A_0\delta x_{aqm} + B_0\delta q \tag{15}$$

and (see Appendix B)

$$2\sqrt{2p_0} = \left[\frac{dC}{d\theta}x_{aqm} + \frac{dD}{d\theta}(q - q_{ref})\right]\Big|_0 \tag{16}$$

$$0 = \left[\frac{dA}{d\theta}x_{aqm} + \frac{dB}{d\theta}(q - q_{ref})\right]\Big|_0 \tag{17}$$

From (16), (14) and (13), we have

$$\delta p = 2\sqrt{2p_0}\delta\theta + C_0\delta x_{aqm} + D_0\delta q \tag{18}$$

$$\delta\dot{\theta}^n_{rc} = \frac{K}{2\sqrt{2p_0}}(C_0\delta x_{aqm} + D_0\delta q) \tag{19}$$

$$\tag{20}$$

From (17) and (15), we have

$$\delta\dot{x}_{aqm} = A_0\delta x_{aqm} + B_0\delta q \tag{21}$$

Then, from (10),(11),(18),(19),(21), we are able to draw the control block diagram for this general STAQM in Figure 10. It is obvious that Figure 10 can be reduced to Figure 9.

We note that the filter time constant $\frac{1}{K_c}$ associated with the estimate of link capacity does not enter into our local stability analysis. Second, the local stability of (7) can be expressed in terms of the stability of a feedback connection between $\frac{K^n_{rc}}{s}$ and

$$T(s) \triangleq \frac{P(s)C_{aqm}(s)e^{-sR}}{1 + P(s)C_{aqm}(s)e^{-sR}}.$$

In context of the adaptive control problem, the transfer function $T(s)$ represents the closed-loop dynamic of a perfectly tuned TCP-AQM loop. That is, one where $C_{aqm}(s)$ is tuned based on perfect estimates of the network parameters. Hence, it is a good assumption that $T(s)$ is stable, which means that its poles are confined to the open left-half of the complex plane. We have the following stability result for this feedback connection.

Theorem 1: *Suppose $P(s)C_{aqm}(s)$ is strictly proper, $P(0)C_{aqm}(0) > 0$ and $T(s)$ is stable. Then, (7) is locally exponentially stable for some sufficiently large and positive time constant $\frac{1}{K^n_{nc}}$.*

Proof: The adaptive system (7) is locally exponentially stable provided the closed-loop system in Figure 9 is stable; i.e.,

$$1 + K^n_{rc}\frac{T(s)}{s} = 0 \tag{22}$$

has solutions only in the open left-half of the complex plane (OLHP). We claim this is so for some sufficiently small and positive K^n_{rc}. We will use root locus arguments to prove this; see [15]. Since $C_{aqm}(s)$ is proper, than $\frac{T(s)}{s}$ is strictly proper. Also, $T(s)$ has only poles in the open left-half of the complex plane.[4] Thus, for sufficiently small root

[4]Due to the delay term e^{-sR}, $T(s)$ has an infinite number of poles. Since these poles are determined by the zeros of a delay polynomial with *principal part*, these poles are strictly bounded away from the imaginary axis of the complex plane.

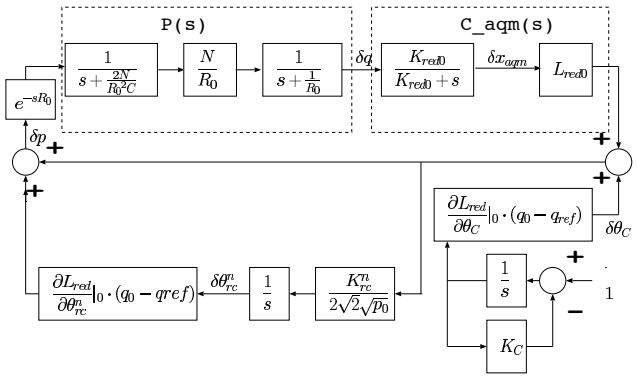

Fig. 7. Linearized Model of STRED.

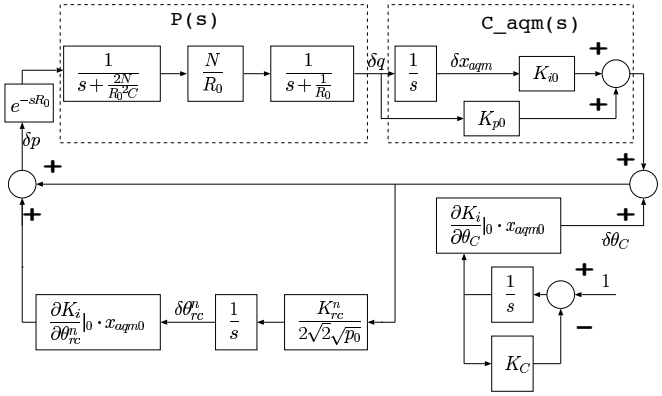

Fig. 8. Linearized Model of STPI.

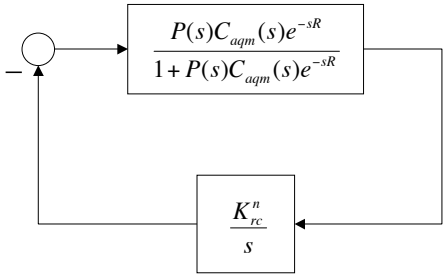

Fig. 9. Feedback loop associated with the local stability of (7). The upper transfer function represents the closed-loop dynamic of a perfectly tuned TCP-AQM loop.

locus parameter K_{rc}^n, the solutions to (22) are in the OLHP, except possibly for a small real solution emanating from the pole of $\frac{T(s)}{s}$ at $s = 0$. Let this solution be $s = \epsilon$. It must be negative; i.e., $\epsilon < 0$. To see this, evaluate (22):

$$1 + K_{rc}^n \frac{T(\epsilon)}{\epsilon} = 0.$$

Since $T(\epsilon) > 0$ (recall $T(0) > 0$), then ϵ is negative for $K_{rc}^n > 0$. We have thus shown that (22) has solutions only in the OLHP when K_{rc}^n is sufficiently small and positive. This proves the theorem. □

Remark: *The implication in Theorem 1 is that small adaption time constants can lead to local instability. On the other hand, excessively large time constants compromise adaptability. Giving quantitative meaning to this tradeoff requires more description of the AQM dynamic $C_{aqm}(s)$. For example, consider the PI AQM in (3) and (7). Then, $T(s)$ is given by*

$$T(s) = \frac{\frac{1}{R^2}\beta_{pi}|j\beta_{pi} + 1|e^{-sR}}{s^2 + \frac{1}{R}s + \frac{1}{R^2}\beta_{pi}|j\beta_{pi} + 1|e^{-sR}},$$

and the range of K_{rc}^n for which (7) is locally stable can be found from those K_{rc}^n rendering Figure 9 stable, or

$$\frac{1}{1 + \frac{K_{rc}^n}{s}T(s)}$$

stable. In Figure 11 we plot the stabilizing time constants $\frac{1}{K_{rc}^n}$ as a function of the aggressiveness parameter β_{pi} and round-trip time R. Stabilizing $\frac{1}{K_{rc}^n}$ lie above the curves. For example, for $R = 100$ ms and $\beta_{pi} = 0.5$, (7) is stabilized for time constants greater than 300ms. [5] *We also see in this figure that larger adaptation (or stabilizing) time constant*

[5] For $\beta_{pi} = 0.5$, the AQM time constant is approximately $\frac{R}{\beta_{pi}} = 200$ms.

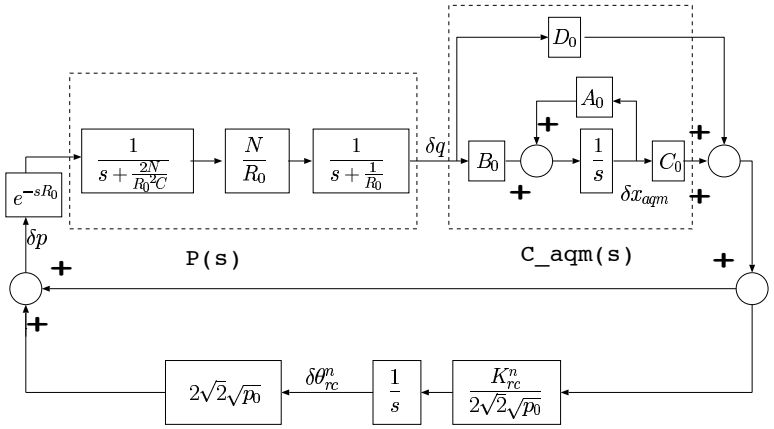

Fig. 10. Linearized Model of A General STAQM.

could result a less adaptive system. Thus, this tradeoff should be considered when designing a STAQM. Similar results could be provided for STRED; see Appendix C.

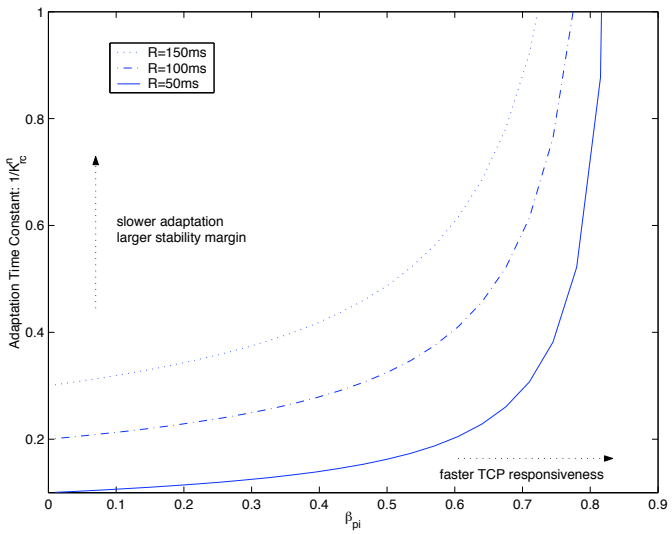

Fig. 11. STPI. Stabilizing time constants $\frac{1}{K_{rc}^n}$ are plotted against the PI aggressiveness factor β_{pi} for various round-trip times R. Stabilizing pairs lie above the curves.

E. Design Guidelines for STRED

Designing STPI is straightforward, however, additional issues arise when designing STRED. The issue is that a stabilizing RED AQM is sufficiently responsive for a specified range of network parameters (N, R, C). From [11], we know that a stablizing RED requires $\omega_g << \min\left\{\frac{2N}{R^2C}, \frac{1}{R}\right\}$ where $1/\omega_g$ is roughly the closed-loop time constant. Clearly, this time constant depends on network parameters (N, C, R). For a fixed round-trip time, smaller ratios N/C result in sluggish AQM control as shown in Figure 12. For example, when $R = 0.15$ sec and $N/C = 0.0005$, the time constant of a stable RED AQM is more than 50 seconds! Indeed, in the next section, we consider simulations where ratios of $N/C = 100/155500 = 0.0006$ (with time constant of 40 seconds) lead to impractical RED designs. Consequently, we will not design STREDs for such scenarios. Instead, we would recommend a proportional controller as in [11], achieved by simply removing RED's low pass filter. Now, for the range of network parameters in which RED has reasonable responsiveness, designing a self-tuning RED (STRED) is straightforward. Details can be found in Appendix C.

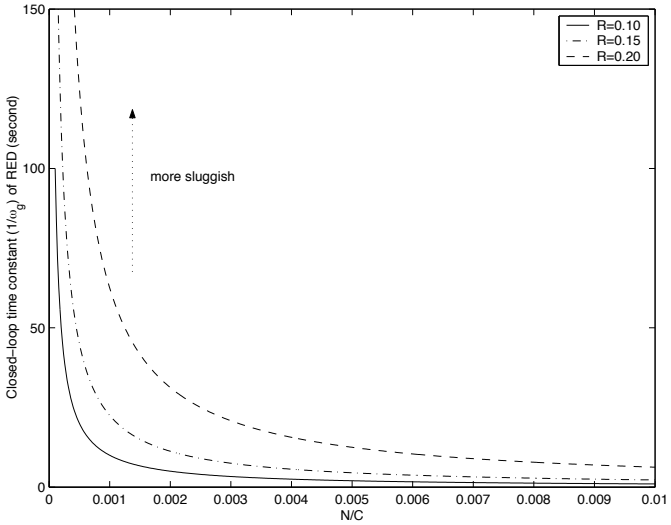

Fig. 12. Fixed RED. Responsiveness of a stable RED as a function of (N, C, R)

IV. ns Simulations

Four ns simulations were conducted to evaluate the performance of the self-tuning AQMs, STPI and STRED, against fixed PI and ARED.

A. *Simulation 1. (Compare PI, ARED, STRED, STPI for variations in link capacity C.)*

The topology for simulation 1 is shown in Figure 13 and consists of TCP and UDP traffic sharing a single congested (OC-12) link with capacity of 622 Mbps (155500 packets/sec). The TCP and UDP traffic have equal priority in competing for link capacity and the AQM does not distinguish between them when dropping or marking packets. There are 1000 long-lived TCP flows with round-trip times uniformly distributed in $(40, 250)$ms. TCP Sack is used and the simulation lasts 1000 seconds. Besides the shared queue simulations presented in this paper, we have also considered class-based queuing (CBQ) links where UDP flows have higher priority than TCP flows.

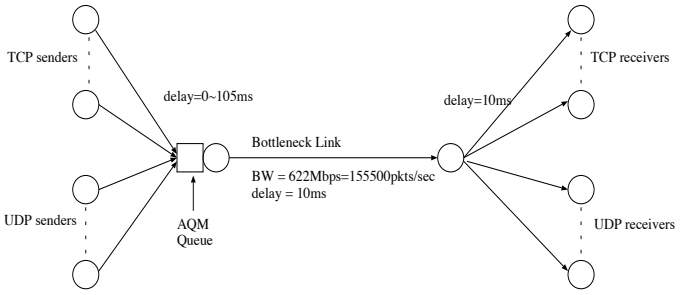

Fig. 13. Simulation 1. Topology used in simulations.

Variations in the UDP traffic are used to generate link capacity variations as experienced by TCP traffic. The UDP traffic commences at 100 seconds, with individual flows arriving according to a Poisson process having sinusoidal arrival with a 400 second period. Each flow generates a number of "on" periods, and this number is distributed geometrically with a mean of 50. For each flow, the "on" period follows a Pareto distribution with mean of 0.2 seconds, while the "off" period is exponentially distributed with a mean of 0.2 seconds. The UDP flow's sending rate is 20Mbps during its "on" period. Due to the difficulty in simulating a large population of UDP sources with small peak rates we simulated a small number of UDP sources having 20Mbps peak rate. Four AQMs are tested: STPI, STRED, PI and ARED. ARED was designed based on the recommendations in [2] and, for comparisons sake, the free

parameters of STPI, STRED and PI were then set to achieve comparable target queue lengths. The following gives the details in setting these parameters.

ARED: Since the link capacity is uncertain, the queue-averaging parameter w_q is set conservatively using the full link capacity 622 Mbps or 155500 packets/second. This gives $w_q = 1 - e^{-\frac{1}{C}} = 6.4308 \times 10^{-6}$. Because ARED automatically chooses 5ms as the target queuing delay, we also take $delay_{target} = 5$ms, $min_{th} = \max\{5, delay_{target}C/2\} = 400$ packets and $max_{th} = 3min_{th} = 1200$ packets. All other parameters of ARED are taken as default.

STPI: The target queue length (q_{ref}) is chosen as 800 packets and parameters a and b are initially designed for full link capacity ($C = 622$ Mbps) and $N = 1000$. Parameter estimation begins immediately and AQM self-tuning is enabled at 20 seconds. The adaptation time constants are $K_C = 5$ seconds and $K_{rc}^n = 10$ respectively. Note that time constant K_{rc}^n falls within the stable region of Figure 11.

STRED: The target queuing delay is taken as $delay_{target} = 5$ms. Following the design guidelines in the Appendix, STRED is designed for $C = 622$ Mbps (full link capacity) and $N = 1000$. For $R = 0.15$ and $N/C = 0.006$ we use Figure 12 and take the adaptation time constant $1/K_{rc}^n$ to be about 5 second. This is the parameter regime for which we believe STRED is meaningful. The RED parameters are $w_q = 0.00000024056$, $max_p = 0.0074$, $min_{th} = 0$, $max_{th} = 3000$. Parameter estimation begins immediately and AQM self-tuning is enabled at 20 seconds. The LPF time constants for the estimates of C and $\frac{N}{RC}$ are 5 and 10 seconds respectively. These time constants fall in the stable region of the plot for $\frac{N}{C} = 0.006$ in Figure 29.

PI: The target queue length (or q_{ref}) is set at 800 packets. Parameters a and b are designed for the full-link capacity of $C = 622$ Mbps and $N = 1000$. This is a conservative PI design, where we expect the PI AQM to become less responsive (rather than unstable) when the link capacity drops.

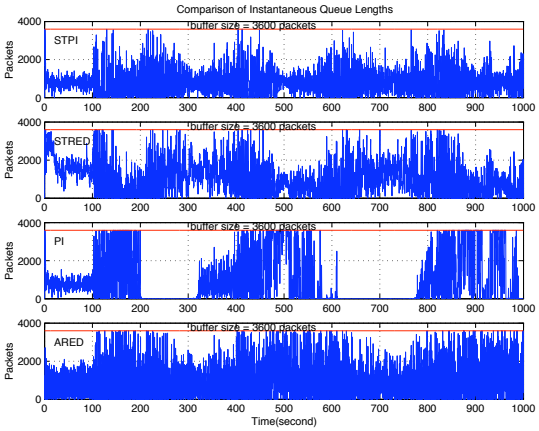

Fig. 14. Simulation 1. Comparison of instantaneous queue lengths for STPI, STRED, fixed PI, and ARED.

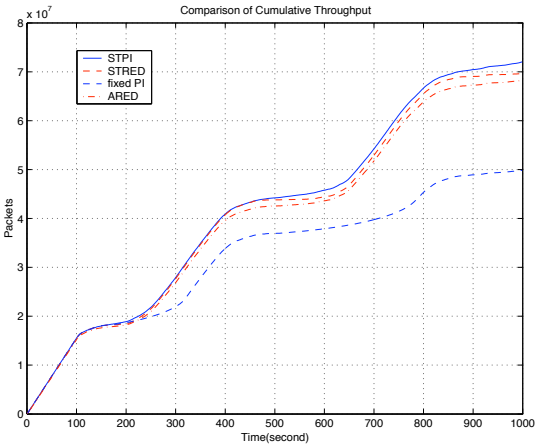

Fig. 15. Simulation 1. Comparison of cumulative throughput.

Figure 15 compares the cumulative TCP throughput (departure rate of TCP packets in the AQM controlled queue) where STPI has the highest throughput, approximately 5% higher than that of ARED at 1000 seconds. STRED's thoughput is between STPI and ARED. The fixed PI's throughput is about 45% less than that of STPI.

Figure 14 shows the instantaneous queue length at 0.1 second intervals. Figure 16 shows the instantaneous queue length when the link capacity experienced by TCP flows is high (about 135000 packets/second). We see that STPI regulates the queue length around the target queue length of 800 packets. STRED has similar performance, and as predicted, the instantaneous queue length of STRED is controlled in the range of min_{th} and max_{th}. On the other hand, the queues controlled by fixed PI and ARED are often empty. When the link capacity is low, STPI regulates

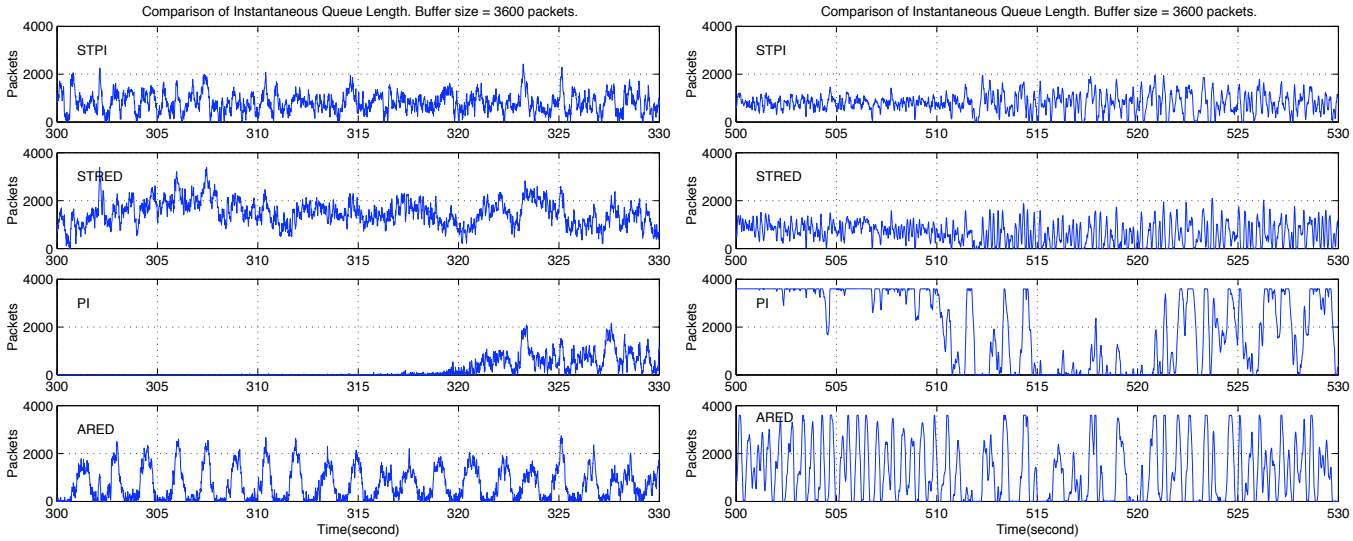

Fig. 16. Simulation 1. Comparison of instantaneous queue lengths for STPI, STRED, fixed PI, and ARED. The left plot is over time interval (300, 330) seconds when TCP experiences highest link capacity (about 135000 packets/second). The right plot is in time interval (500, 530) seconds where TCP flows experience lowest link capacity (about 17000 packets/second).

the queue length, whereas ARED and fixed PI oscillate between an empty buffer and its limit of 3600 packets); see Figure 16. As for queuing delays, recall that the AQM designs were driven by ARED's choice to regulate delay around a 5 ms target. Consistent with their regulation of instantaneous queue length, STPI and STRED regulate the queuing delay around 5 ms, whereas, ARED and fixed PI have significant jitter. The standard deviation of queuing delay for ARED is 8.1ms, almost twice that of STPI's 4.3ms. STRED is 5.8ms. Fixed PI's is 6.5ms. In summary, STAQMs(STPI and STRED) are able to regulate the queuing delays with much less variation than fixed PI and ARED. Also, the STAQMs achieved the highest throughput.

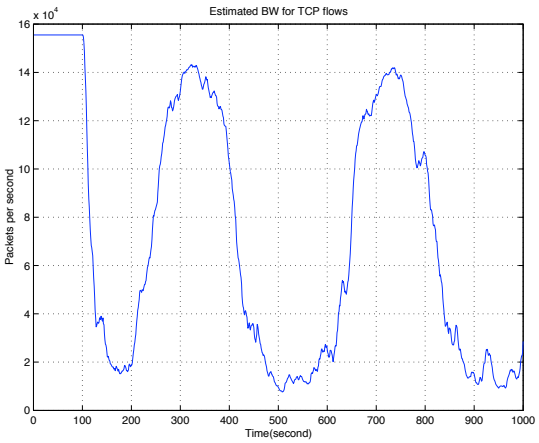

Fig. 17. Simulation 1. Link capacity (estimated by STPI) experienced by TCP flows.

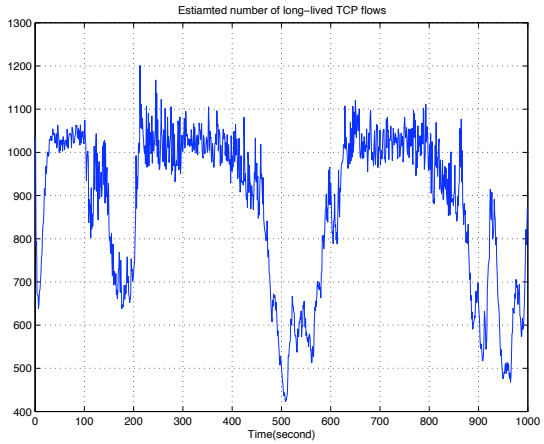

Fig. 18. Simulation 1. Number of TCP flows implicitly estimated by STPI.

Finally, in Figure 17 we plot the estimated link capacity and in Figure 18, the estimate of number of TCP flows N. As explained in Section III.B.2, our techniques underestimate N, as confirmed in this figure.

B. Simulation 2. (Compare PI, ARED, STPI for variations in TCP workload N.)

We use the same topology as in the preceding simulation but do not include UDP traffic; thus, the link capacity experienced by TCP flows is fixed at 622 Mbps. We allow the number of long-lived TCP flows vary between 100 and 1000 flows as shown in Figure 19. Again, we use heterogeneous round-trip times distributed over the 40 - 250 ms

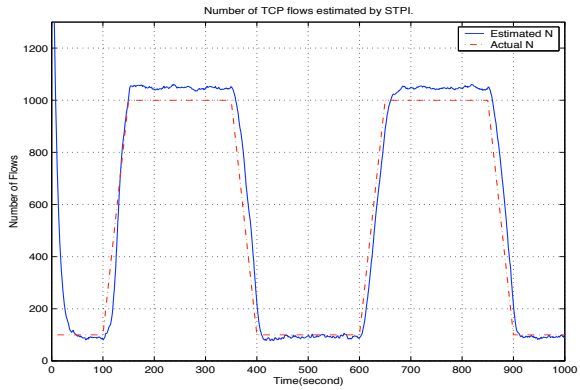

Fig. 19. Simulation 2. The number of TCP flows estimated by STPI.

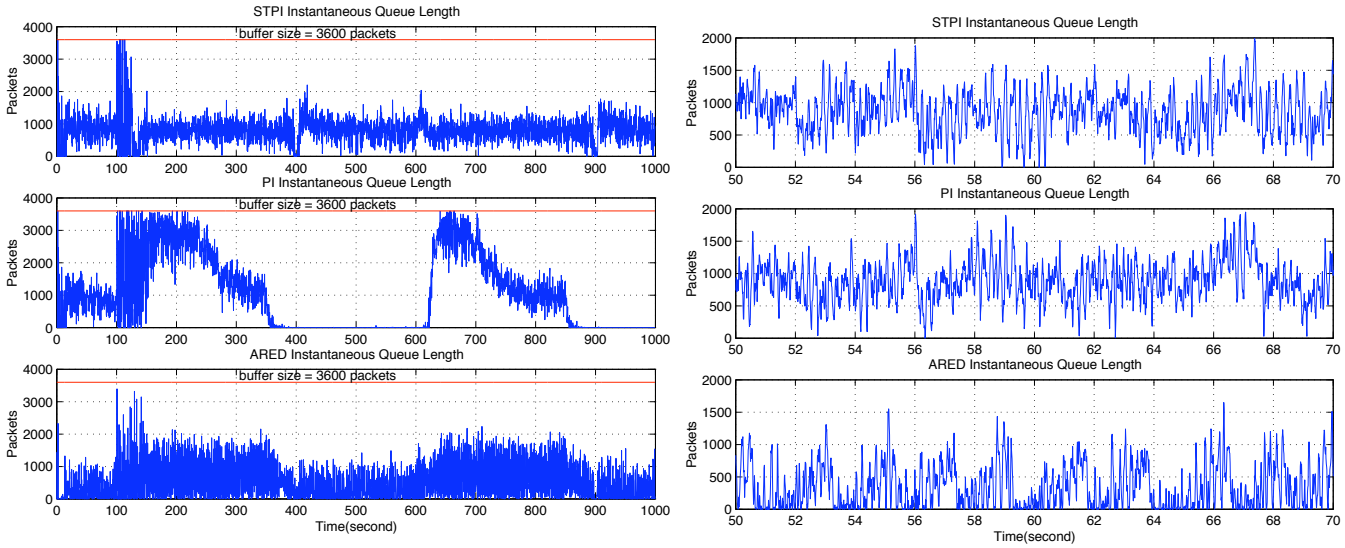

Fig. 20. Simulation 2. Comparison of instantaneous queue lengths for STPI, fixed PI, and ARED. The left plot is over whole simulation duration. The right plot is in time interval (50, 70) seconds where TCP load level is lowest (100 active TCP flows.)

range. Since the ratio N/C could be as low as 0.0006, a stabilizing RED controller would be very sluggish as discussed in Section 3. Thus, we will not design a STRED for these conditions and compare only the performance of STPI, fixed PI and ARED. Since ARED's parameters do not explicitly depend on TCP load, we use the same parameters as in Simulation 1. The target queue length is between 400 and 1200 packets. The fixed PI is conservatively designed for an initial load level $N = 100$ flows and full link capacity $C = 622$ Mbps. Its target queue length is 800 packets.

As shown in Figure 20, STPI regulates the queue length to its target level; but fixed PI is slow to respond when N increases from 100 to 1000 flows from 100 to 300 seconds. Also, there is a long period when the queue is empty when N decreases from 1000 to 100 from 900 to 1000 seconds. ARED oscillates between an empty queue and large queue length of 2000 packets – this occurs under high load $N = 1000$. This behavior is similar to the situation in the left plot of Figure 16. STPI controls the queuing delay to the 5ms target delay, while the queuing delay for ARED oscillates between 0 and 10 ms (although its target delay is also 5ms). The standard deviation of queuing delay for ARED is 7.1ms, almost three times that of STPI's 2.5 ms. Fixed PI's has standard deviation of queuing delay of 3.5ms. In summary, STPI was able to regulate queuing delay with much less variation than that of fixed PI and ARED. The throughput for STPI and ARED were very similar.

Finally, we would like to point out a stability issue associated with ARED. TCP fluid models ([11]) show that the steady-state loss rate p must relate to network conditions as $p = \frac{2N^2}{R^2C^2}$. For the set-up in simulation 2 this means that the steady-state p should approximately be 0.000036761. However, as shown in the left plot of Figure 21, ARED

does *not* achieve the expected steady-state loss but produces much more. Furthermore, ARED appears to be locally unstable. Following the notation in [11], the transfer function for RED is $C(s) = \frac{K_{red}L_{red}}{s+K_{red}}$. ARED is essentially RED with K_{red} fixed to 1 second and where L_{red} varies with max_p which, in turn, is tied to average queue length. The range of max_p is $(0.01, 0.5)$. Thus, when $N = 100, C = 155500, R = 0.15, min_{th} = 400$ and $max_{th} = 1200$, ARED's L_{red} varies in the range $(1.2500e-05, 6.2500e-04)$. Analysis indicates that this range of L_{red} leads to local instability and helps to explain ARED's oscillation as shown in the right plot of Figure 20. In contrast, the average loss rate p for STPI is about 0.00005, close to that required by fixed-point analysis. Compare the plots on the right-hand side of Figure 20 in the time interval $(100, 200)$ with the left plot of Figure 5. The latter case has a homogeneous RTT of 150ms, with the former case having heterogenous RTTs with an average of 150ms. The heterogenous RTTs appear to mitigate (but not quench) the oscillations in ARED.

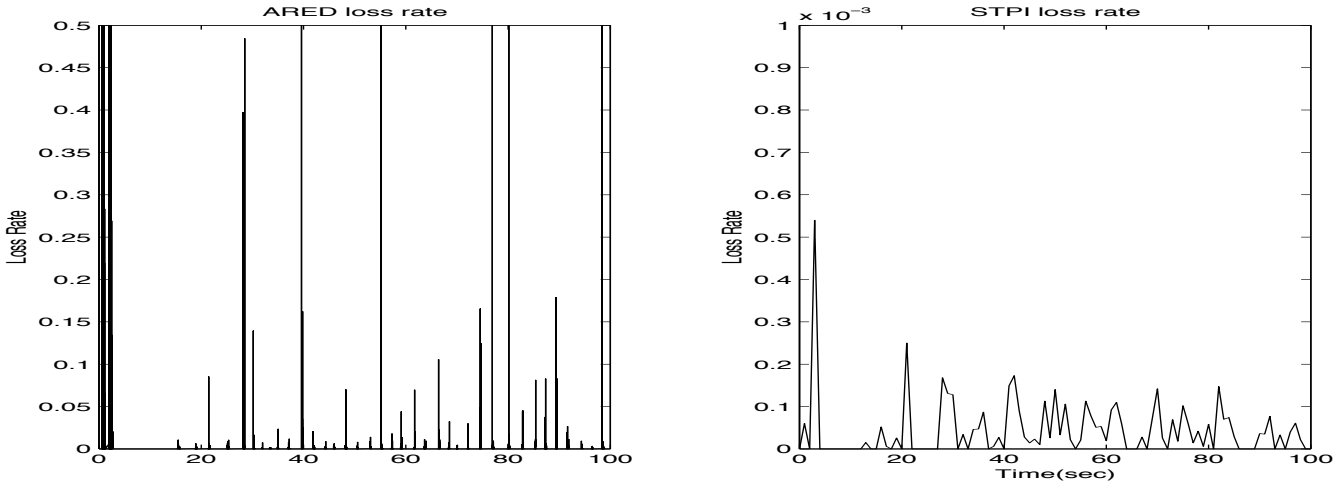

Fig. 21. Simulation 2. Loss rate comparison between ARED and STPI for the smallest load (N=100).

C. Simulation 3. (Compare PI, ARED, STPI, STRED for variations in link capacity C for Web traffic and reverse long-lived TCP traffic.)

In our third simulation, we consider a richer mix of traffic which contains long-lived TCP flows and short-lived TCP flows (https flows). Figure 22 shows the simulation topology. There are 1000 long-lived TCP flows traveling in both the forward and reverse paths. The round trip delays for the long-lived flows are uniformly distributed in (40, 250)ms. In addition, there are 4000 http flows on the forward path, and 4000 http flows on the reverse path. Each http flow idles after finishing a session. The idle time is an exponential random variable with mean of 3 seconds. Two AQMs control both the forward and reverse path queues. The simulation lasts 500 seconds.

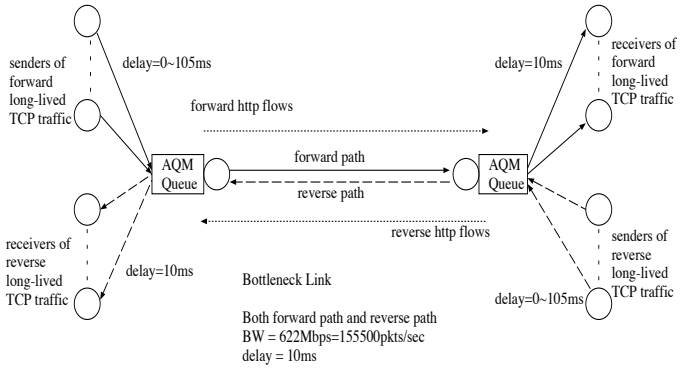

Fig. 22. Simulation 3. Topology used in simulations.

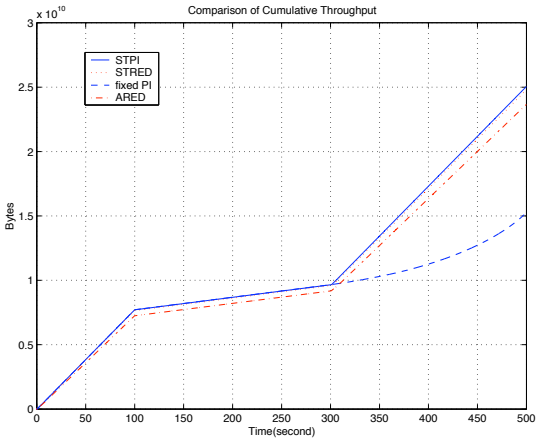

Fig. 23. Simulation 3. Comparison of accumulative throughput among STPI, STRED, fixed PI, and ARED.

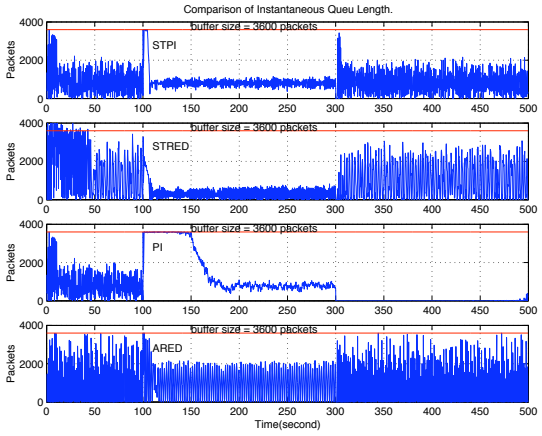

Fig. 24. Simulation 3. Comparison of instantaneous queue lengths for STPI, STRED, fixed PI, and ARED.

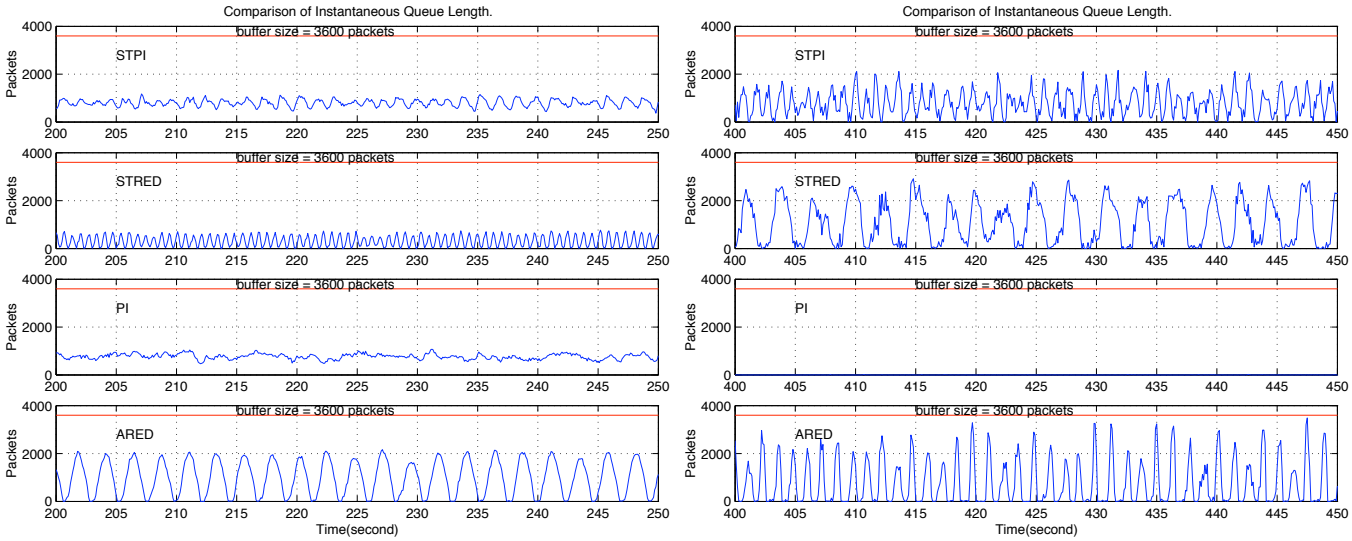

Fig. 25. Simulation 3. Comparison of instantaneous queue lengths for STPI, STRED, fixed PI, and ARED. The left plot is in time interval (200, 250) seconds where link capacity is lowest (77.75Mbps or 19438 pkts/sec.) The right plot is in time interval (400, 450) seconds where link capacity is highest (622Mbps or 155500 pkts/sec.)

We consider a scenario of varying link capacity in the forward path. Initially, the link capacity experienced by both forward and reverse traffic is 622Mbps. At 100 seconds, we decrease the link capacity by a factor of 8, then, we return to 622Mbps at 300 seconds. The link capacity of the reverse path is kept constant at 622Mbps. For all four AQMs, we use the same parameters as in Simulation 1. The target queue length is between 400 and 1200 packets for ARED, and the target queue length for STPI and fixed PI are 800 packets. The size of TCP data packets is 1000 bytes. Since the ACK packets (40 bytes per packet) share the queue with TCP data packets, we set the mean packet size to be 500 bytes. Since the queued packets have different sizes, we used the mean packet size. Similar to Simulation 1, for a stable RED in this scenario, the target queue length is not a fixed value, but varies between 0 and 3000 pkts. The link capacity estimated by STPI is quite accurate, and the traffic load is underestimated as before. Figure 23 compares the accumulative throughput where STPI has the highest throughput at 500 seconds, approximately 6% higher than that of ARED, and 64% percent higher than fixed PI. STRED has almost the same throughput as STPI. Figure 24 compares instantaneous queue lengths. STPI regulates the queue length well while ARED and fixed PI oscillate. These oscillations affect queuing delay. Regardless of network conditions, STPI regulates the queuing delay with little jitter. The standard deviation of queuing delay for STPI, STRED, fixed PI and ARED are 3.5ms, 6.1ms, 7.0ms, and 5.8ms respectively. Similar to the previous simulations, STPI is able to regulate queuing delay with much less variation

than that of fixed PI and ARED, and achieved the highest throughput. Even though STRED's throughput is almost the same as that of STPI, STRED's queuing delay can *not* be guaranteed.

D. Simulation 4. (Compare PI, ARED, STPI for variations in TCP workload N for Web traffic and reverse long-lived TCP traffic.)

In this simulation, we also consider a richer mix of traffic which contains long-lived TCP flows and short-lived TCP flows (https flows). We use the same simulation topology as in simulation 3. We keep the link capacity constant at $622Mbps$, and vary the number of long-lived TCP flows. Initially, there are 200 long-lived TCP flows travelling in both the forward and reverse paths. The round trip delays for the long-lived flows are uniformly distributed in (40, 250)ms. In addition, there are 800 http flows on the forward path, and 800 http flows on the reverse path. Each http flow idles after finishing a session. The idle time is an exponential random variable with mean of 3 seconds. Two AQMs control both the forward and reverse path queues. At 200 seconds, an additional 800 long-lived TCP flows and 3200 http flow in both directions. At 400 seconds, these newly added long-lived TCP flows drop out. The simulation lasts 500 seconds.

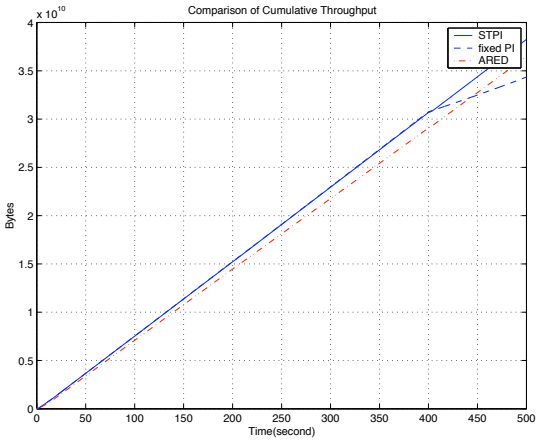

Fig. 26. Simulation 4. Comparison of accumulative throughput among STPI, fixed PI, and ARED.

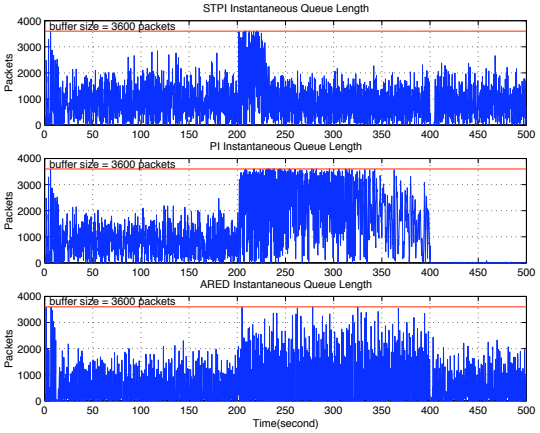

Fig. 27. Simulation 4. Comparison of instantaneous queue lengths for STPI, fixed PI, and ARED.

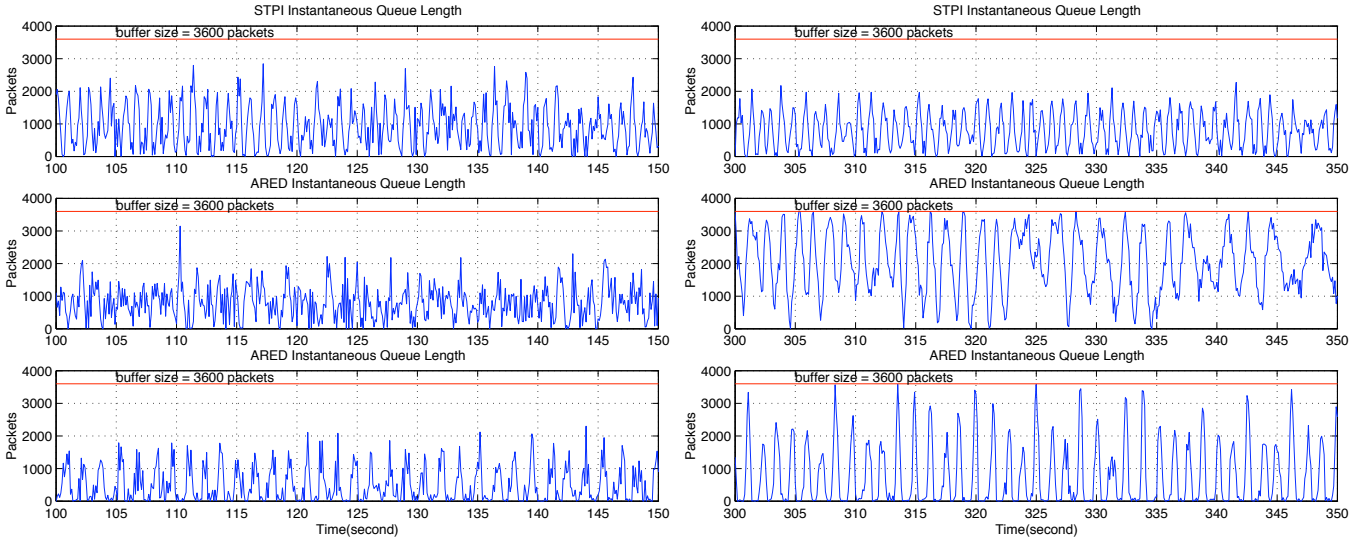

Fig. 28. Simulation 4. Comparison of instantaneous queue lengths for STPI, STRED, fixed PI, and ARED. The left plot is in time interval (100, 150) seconds where the number of flows is smallest (200 long-lived TCP flows). The right plot is in time interval (300,350) seconds where the number of flows is highest (1000 long-lived TCP flows)

We do not consider a STRED because the lowest value of $\frac{N}{C}$ is 0.0013 render stable RED to be very sluggish as shown in Figure 12. For all three AQMs (STPI, PI, ARED), we use the same parameters as in Simulation 1. The target queue length is between 400 and 1200 packets for ARED, and the target queue length for STPI and fixed PI are 800 packets. The size of TCP data packet is 1000 bytes. Similar to Simulation 3, we set the mean packet size to 500 bytes, and we measured the queue length in bytes. Fixed PI is designed for N=200.

As in simulation 3, the link capacity estimated by STPI is quite accurate, and the traffic load is underestimated. Figure 26 compares the accumulative throughput where STPI achieves the highest throughput at 500 seconds, approximately 4.89% higher than that of ARED, and 11.35% percent higher than fixed PI. Figure 27 compares instantaneous queue lengths. STPI regulates the queue length well, on the contrary, ARED and fixed PI oscillate, as shown in figure 28. Regardless of network condition change, STPI is able to regulate the queuing delay and with little jitter. These oscillations affect queuing delay. The standard deviations of queuing delays for STPI, fixed PI and ARED are 4.4ms, 4.8ms, and 6.8ms respctively. Similar to the previous simulations, STPI was able to regulate queuing delay with much less variation than that of fixed PI and ARED, and achieved the highest throughput.

E. General Discussions on Simulations

From these simulations we see that the STAQM structure (especially STPI) provides robust performance in the face of network parameter variations. STPI and STRED achieve the highest cumulative throughput; see Figure 15, Figure 23, and Figure 26 in comparison to ARED and fixed PI. Also, STPI regulates the queuing delay to the target delay with the smallest jitter. It quickly adapts to large changes in link capacity and traffic load level, maintaining control of queue length. This latter performance distinguishes STPI from ARED. In Simulation 2, ARED's variation queue length variation is much larger.

Another important point to note, is that use of STRED (as well as RED) is limited to the larger ratios N/C as shown in Figure 12. For the N/C ranges corresponding to low sluggishness, STRED works well by regulating the queue length in the range of minimum and maximum queue length, as shown in Simulation 1 and Simulation 3. If we modify RED to be a proportional controller (achieve this by removing it averaging filter completely), then STRED is applicable under any network condition. But, there is a caveat. AQMs that do not employ an integrator, like RED and proportional, have equilibrium queue lengths that scale with link capacity. Now, for RED and ARED, the tricky implication of this is to insure this equilibrium falls within the user-specified range. Indeed, as shown in Figure 20, the suggested ARED settings lead to an unstable queue. ARED forces the queue length to stabilize in a range that does not include its equilibrium value! This leads to bad oscillations.

In summary, these `ns` simulations confirm robust performance of STAQM over a wide range of network parameters and scenarios that included a rich traffic mix including Web and reverse long-lived TCP sessions.

V. Conclusions

The self-tuning structure proposed in this paper is straightforward to implement and is applicable to any AQM scheme that can be parameterized in terms of link capacity and TCP load. When linked with a PI AQM, the resulting STPI adaptive controller is, as far as `ns` simulations indicate, remarkably robust to variations in link capacity and TCP load. This adaptability is crucial since in practice network conditions vary a lot, and can lead to unstable or sluggish controllers with a static designs. STPI compares favorably with ARED in terms of throughput and queuing delay, and, can deal with high-capacity links. Future research directions include more sophisticated stability analysis, expanded parameter estimation, and application to larger and more realistic networks. We are also exploring a sampling based estimator for the "effective" RTT of the flows at a congested link, enabling an even better performing self-tuning controller.

References

[1] S. Floyd and V. Jacobson, "Random Early Detection gateways for congestion avoidance," *IEEE/ACM Transaction on Networking*, vol. 1, no.4, August 1997

[2] S. Floyd and R. Gummadi and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED," Technical Report, to appear, 2001.
[3] F. Kelly, "Mathematical Modeling of the Internet," *Mathematics Unlimited - 2001 and Beyond*, 2000.
[4] S. H. Low, F. Paganini, and J.C. Doyle, "Internet Congestion Control," *IEEE Control Systems Magazine*, Vol. 22, no. 1, pp. 28-43, 2002.
[5] W. Wu, Y. Ren, and X. Shan, "A Self-configuring PI Controller for Active Queue Management," in *Asia-Pacific Conference on Communications (APCC)*, Session T53, Tokyo, Japan, 2001.
[6] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Blue: A new Class of Active Queue Management Algorithms," Tech. Rep., UM CSE-TR-387-99, 1999.
[7] W. Feng, Dilip D. Kandlur, Debanjan Sahar, and Kang G. Shin, "A Self-Configuring RED Gateway," in *Proceedings of IEEE/INFOCOM*, 1999
[8] Vishal Misra, Wei-Bo Gong, and Don Towsley, "Fluid-based Analysis of a Network of AQM Routers Supporting TCP flows with an Application to RED," in *Proceedings of ACM/SIGCOMM, 2000.*
[9] C.V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," in *Proceedings of IEEE/INFOCOM*, April 2001.
[10] C.V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong, "A Control Theoretic Analysis of RED," in *Proceedings of IEEE/INFOCOM*, April 2001.
[11] C.V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong, "Analysis and Design of Controllers for AQM Routers Supporting TCP Flows," in *IEEE TAC's special issue on Systems and Control Methods for Communication Networks*, Vol. 47, no. 6, 2002.
[12] S. Kunniyur and R. Srikant. "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," in *Proc. of ACM SIGCOMM 2001*, August 2001.
[13] Yuan Gao, Guanghui He, and Jennifer Chao-Ju Hou. "On leveraging traffic predictability in active queue management," in *Proceedings of IEEE/INFOCOM*, June 2002.
[14] K.J. Astrom, B. Wittenmark, "Adaptive Control," Addison-Welsley. Reading, Massachusetts, 1995.
[15] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*. Addison-Wesley. Reading, Massachusetts, 1995.

APPENDIX

A. Linearization of Self-Tuning AQMs (STRED and STPI)

The following equations describe the self-tuned TCP/AQM dynamic.

$$\begin{aligned}
\dot{W} &= \frac{1}{R} - \frac{W^2}{2R} p_R \\
\dot{q} &= \frac{N}{R} W - C; \\
\dot{\theta}_c &= -K_c \theta_c + K_c \hat{C}; \\
\dot{\theta}^n_{rc} &= -K^n_{rc} \theta^n_{rc} + K^n_{rc} \sqrt{p/2}; \\
\dot{x}_{aqm} &= f_\theta(x_{aqm}, q) = \begin{cases} -K_{red} x_{aqm} + K_{red} q & \text{STRED} \\ q - q_{ref} & \text{STPI} \end{cases} \\
p &= g_\theta(x_{aqm}, q) = \begin{cases} L_{red}(x_{aqm} - q_{ref}) & \text{STRED} \\ K_i x_{aqm} + K_p(q - q_{ref}) & \text{STPI} \end{cases}
\end{aligned} \tag{23}$$

State variables are: $(W, q, \theta_C, \theta^n_{rc}, x_{aqm}, p)$. We will show the linearizations of the above equations at the operating points of state variables.

First, the operating point relationships are:

$$\begin{aligned}
\dot{W} = 0 &\rightarrow W_0^2 p_0 = 2 \\
\dot{q} = 0 &\rightarrow W_0 = \frac{R_0 C}{N} \\
\dot{\theta}_c = 0 &\rightarrow \theta_{C0} = C \\
\dot{\theta}^n_{rc} = 0 &\rightarrow \sqrt{\frac{p_0}{2}} = \frac{1}{W_0}
\end{aligned}$$

$$\dot{x}_{aqm} = 0 \quad \rightarrow \quad \begin{cases} x_{aqm0} = q_0 & \text{STRED} \\ q_0 = q_{ref} & \text{STPI} \end{cases}$$
$$p_0 = g_{\theta 0}(x_{aqm0}, q_0) \quad \rightarrow \quad \begin{cases} p_0 = L_{red0}(q_0 - q_{ref}) & \text{STRED} \\ x_{aqm0} = \frac{p_0}{K_{i0}} & \text{STPI} \end{cases} \tag{24}$$

where $R_0 = T_p + \frac{q_0}{C_0}$.

Evaluating the partials of those functions at the operating points gives:

$$\begin{aligned}
\delta\dot{W}(t) &= -\frac{2N}{R_0^2 C}\delta W - \frac{R_0^2 C^2}{2N^2}\delta p_R \\
\delta\dot{q}(t) &= \frac{N}{R_0}\delta W - \frac{1}{R_0}\delta q(t) \\
\delta\dot{\theta}_c &= -K_C \delta\theta_C \\
\delta\dot{\theta}^n_{rc} &= -K^n_{rc}\delta\theta^n_{rc} + \frac{K^n_{rc}}{2\sqrt{2}\sqrt{p}}\delta p \\
\delta p &= \begin{cases} \left[\frac{\partial L_{red}}{\partial \theta^n_{rc}}(x_{aqm} - q_{ref})\right]\Big|_0 \cdot \delta\theta^n_{rc} + \left[\frac{\partial L_{red}}{\partial \theta_C}(x_{aqm} - q_{ref})\right]\Big|_0 \cdot \delta\theta_C + L_{red0}\cdot\delta x_{aqm} & \text{STRED} \\ \left[\frac{\partial K_i}{\partial \theta^n_{rc}} x_{aqm}\right]\Big|_0 \cdot \delta\theta^n_{rc} + \left[\frac{\partial K_i}{\partial \theta_C} x_{aqm}\right]\Big|_0 \cdot \delta\theta_C + K_{i0}\delta x_{aqm} + K_{p0}\delta q & \text{STPI} \end{cases} \\
\delta\dot{x}_{aqm} &= \begin{cases} -K_{red0}\delta x_{aqm} + K_{red0}\delta q & \text{STRED} \\ \delta q & \text{STPI} \end{cases}
\end{aligned} \tag{25}$$

We will take the derivations of K_i and L_{red}.

$$K_i = \beta_{pi}\sqrt{\beta_{pi}^2 + 1}\cdot 4(\theta^n_{rc})^2 \cdot \frac{1}{R^2}\cdot\frac{1}{\theta_C} = \triangle_{red}\cdot(\theta^n_{rc})^2\cdot\frac{1}{\theta_C} \tag{26}$$

$$L_{red} = \sqrt{\frac{w_g^2}{K_{red}^2} + 1}\cdot\frac{4(\theta^n_{rc})^2}{RC} = \sqrt{1 + tan(\beta_{red} - w_g R)^2}\cdot\frac{4(\theta^n_{rc})^2}{R\theta_C} \tag{27}$$

Since $w_g \ll min\{\frac{2N}{R^2C}, \frac{1}{R}\}$, so $w_g R \ll 1$, then,

$$L_{red} = \sqrt{1 + tan(\beta_{red})^2}\cdot\frac{4(\theta^n_{rc})^2}{RC} = \triangle_{pi}\cdot(\theta^n_{rc})^2\cdot\frac{1}{\theta_C}$$

We notice that K_i and L_{red} have similar forms, by using this property, we can subsitute δp into the expression of $\delta\dot{\theta}^n_{rc}$ to get the following equations:

$$\dot{\theta^n_{rc}} = \begin{cases} -\frac{K^n_{rc}\sqrt{p_0}}{2\sqrt{2}\theta_C}\cdot\delta\theta_C + \frac{K^n_{rc}}{2\sqrt{2}\sqrt{p_0}}\cdot L_{red0}\delta x_{aqm} & \text{STRED} \\ \frac{K^n_{rc}}{2\sqrt{2}\sqrt{p_0}}\cdot\left(K_{i0}\cdot\delta x_{aqm} + K_{p0}\cdot\delta q + \frac{\partial K_i}{\partial\theta_C}\Big|_0\cdot q_0\cdot\delta\theta_C\right) & \text{STPI} \end{cases} \tag{28}$$

B. Two Important Relationships in the Linearization of a Linear Time Invariant Self-Tuning AQM

We now prove the two important relationships in STAQM linearization; namely,

$$2\sqrt{2p_0} = \left[\frac{dC}{d\theta}x_{aqm} + \frac{dD}{d\theta}(q - q_{ref})\right]\Big|_0 \tag{29}$$

$$0 = \left[\frac{dA}{d\theta}x_{aqm} + \frac{dB}{d\theta}(q - q_{ref})\right]\Big|_0 \tag{30}$$

First, we consider (29). We know that

$$p = C(\theta)x_{aqm} + D(\theta)(q - q_{ref}) \tag{31}$$
$$\dot{\theta} = -K\theta + K\sqrt{p/2} \tag{32}$$

From (32), we write p as a function of θ

$$p = \frac{2}{K^2}(\dot{\theta} + K\theta)^2 \tag{33}$$

Take the derivative of (33) gives

$$\frac{dp}{d\theta} = \frac{4}{K}(\dot{\theta} + K\theta) \tag{34}$$

Evaluate (34) at the equilibrium point, then we have

$$\frac{dp}{d\theta}\Big|_0 = 4\theta_0 = 4\sqrt{p_0/2} \tag{35}$$

Take the derivative of (31) gives

$$\frac{dp}{d\theta} = \dot{C}(\theta)x_{aqm} + \dot{D}(\theta)(q - q_{ref}) \tag{36}$$

So from (35) and (36) we have the following relationship

$$2\sqrt{2p_0} = \left[\dot{C}(\theta)x_{aqm} + \dot{D}(\theta)(q - q_{ref})\right]\Big|_0 \tag{37}$$

This completes the proof of (29).

Now, consider $\dot{x}_{aqm} = A(\theta)x_{aqm} + B(\theta)(q - q_{ref})$ with the objective of computing $\frac{\partial \dot{x}_{aqm}}{\partial \theta}$.

Since $\dot{\theta} = 0$ when $\theta = \theta_0$, then [6]

$$\frac{\partial \dot{x}_{aqm}(\theta)}{\partial \theta} = \frac{\partial \dot{x}_{aqm}(\theta_0)}{\partial \theta_0} \tag{38}$$

But,

$$\frac{\partial \dot{x}_{aqm}(\theta_0)}{\partial \theta_0} = \frac{\partial}{\partial \theta_0}(A(\theta_0)x_{aqm} + B(\theta_0)(q - q_{ref})) = 0 \tag{39}$$

This completes the proof of (30).

C. Choice of Time Constant K_{rc}^n for STRED

For a RED AQM, $T(s)$ is given by

$$T(s) = \frac{\frac{2}{R^3} \cdot \frac{N}{C} \cdot \sqrt{tan^2(\beta_{red} - \omega_g R) + 1}}{(s + \frac{2}{R^2} \cdot \frac{N}{C})(s + \frac{1}{R})(\frac{tan(\beta_{red} - \omega_g R)}{\omega_g}s + 1)}$$

The choice of time constant K_{rc}^n is dependent on $\frac{N}{C}$, aggressiveness factor β_{red} and R. We evaluate K_{rc}^n in a range of ($\frac{N}{C}$,R) in Figure 29. As $\frac{N}{C}$ increases, the time constant for a stable system could be smaller. For a given $\frac{N}{C}$, more aggressive (larger β_{red}) RED design needs larger time constant K_{rc}^n.

D. The Procedure for Designing STRED

The following steps should be followed: (i) we solve for the time constant $1/\omega_g$ of a stable RED by choosing a stability phase margin (e.g., 85 degree gives $\beta_{red} = 85 \cdot pi/180$), and choose $\omega_g << min[\frac{2N}{R^2C}, 1/R]$, (e.g., $\omega_g = 0.1 \cdot min[\frac{2N}{R^2C}, 1/R]$); (ii) $K_{red} = \frac{\omega_g}{tan(\beta_{red} - \omega_g * R)}$. Thus, the exponential average weight factor w_q can be decided; (iii) we choose $L_{red} = \frac{|j\omega_g + K_{red}|(2N)^2)}{(K_{red}(RC)^3)}$ to set $|L(j\omega_g)| = 1$; (iv) we will decide (min_{th}, max_{th}) to solve for max_p.

[6] θ can change only on the space of points described by $\theta = \theta_0$

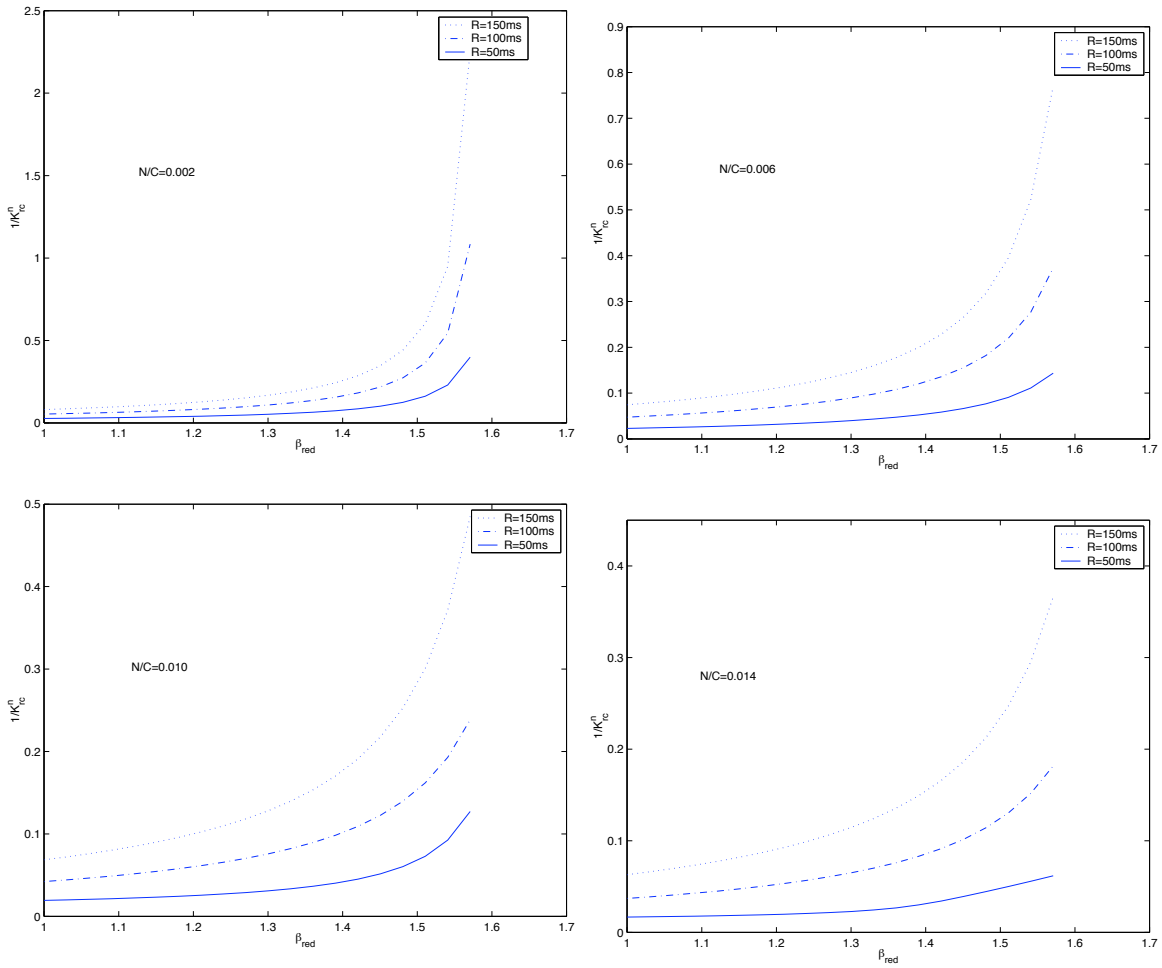

Fig. 29. STRED. Stabilizing time constants are plotted against the RED aggressiveness factor β_{red} for various round-trip times R, and different $\frac{N}{C}$s. Stabilizing values lie above the curves.

In the following analysis, we will show that (min_{th}, max_{th}) cannot be arbitrarily chosen if we want to design a stable RED. Let p_0 and q_0 denote the loss rate and instantaneous queue length in the steady state, W_0 be the window size in steady state. Based on TCP dynamic in the steady state (fluid model in [8]), we know that $p_0 W_0^2 = 2$, and $W_0 = \frac{R_0 C}{N}$, so, $p_0 = \frac{2N^2}{R^2C^2}$. In steady state, q_0 is equal to the average queue length of RED, so, $p_0 = L_{red}(q_0 - min_{th})$ ([11]). Thus, in order to maintain TCP steady state, we need to have $\frac{2N^2}{R^2C^2} = L_{red}(q_0 - min_{th})$, which means $q_0 - min_{th} = \frac{RC}{2\sqrt{1+tan(\beta_{red} - w_g R)^2}}$. Given a target queuing delay, q_0 should be the targe queue length. Let $q_d = q_0 - min_{th}$. If $q_d >= q_0$(chosen by user), then set $min_{th} = 0$ and $q_0 = q_d$. This means that the user specified queuing delay cannot be guaranteed if we are using RED AQM and want a stable system. Otherwise if $q_d < q_0$(chosen by user), we choose $min_{th} = q_0 - q_d$. Then, $max_{th} = q_0 + q_d$ if we want the target queue length to be in the middle of min_{th} and max_{th}. Finally, $max_p = L_{red}(max_{th} - min_{th})$

For example, $(C = 155500, N = 1000, R = 0.15)$ and 5ms queuing delay means $q_0 = 800$. Following the above design rules, we have: $min_{th} = 0, max_{th} = 2300, wq = 0.000000035, max_p = 0.0074$. The actual queuing delay is 0.0072 seconds. This scenario is simulated in the first 100 seconds of Simulation 1.

Note, this design cannot guarantee the queuing delay sepcified by the user, because if the q_d is greater than q_0, in order to maintain RED stability, we have to force q_0 to be q_d and min_{th} to be 0. This is the fundamental property of RED. Note, even though the target queue length chosen by the user can fall in the range of min_{th} and max_{th}. RED cannot guarantee q_0 to be the target queue length due to the nature of a proportional controller.