# Active Queue Management via Event-Driven Feedback Control

Mehmet H. Suzer*, Kyoung-Don Kang, Can Basaran

*Department of Computer Science, State University of New York at Binghamton Binghamton, NY 13902*

## Abstract

Active Queue Management (AQM) is investigated to avoid incipient congestion in gateways to complement congestion control run by the transport layer protocol such as the TCP. Most existing work on AQM can be categorized as (1) ad-hoc event-driven control and (2) time-driven feedback control approaches based on control theory. Ad hoc event-driven approaches for congestion control, such as RED (Random Early Detection), lack a mathematical model. Thus, it is hard to analyze their dynamics and tune the parameters. Time-driven control theoretic approaches based on solid mathematical models have drawbacks too. As they sample the queue length and run AQM algorithm at every fixed time interval, they may not be adaptive enough to an abrupt load surge. Further, they can be executed unnecessarily often under light loads due to the time-driven nature. To seamlessly integrate the advantages of both event-driven and control-theoretic time-driven approaches, we present an event-driven feedback control approach based on formal control theory. As our approach is based on a mathematical model, its performance is more analyzable and predictable than ad hoc event-driven approaches are. Also, it is more reactive to dynamic load changes due to its event-driven nature. Our simulation results show that our event-driven controller effectively maintains the queue length around the specified set-point. It achieves shorter E2E (end-to-end) delays and smaller E2E delay fluctuations than several existing AQM approaches, which are ad-hoc event-driven

*Correspondent Author, Tel: +1 607 761 8934, Fax: +1 607 777 4729
*Email addresses:* `msuzer@cs.binghamton.edu` (Mehmet H. Suzer), `kang@cs.binghamton.edu` (Kyoung-Don Kang), `cbasaran@cs.binghamton.edu` (Can Basaran)

and based on time-driven control theory, while achieving almost the same E2E delays and E2E delay fluctuations as the two other advanced control theoretic AQM approaches. Further, our AQM algorithm is invoked much less frequently than the tested baselines.

*Keywords:* Congestion Control, Active Queue Management, Event-Driven Feedback Control

## 1. Introduction

Congestion control and avoidance is critical. Packets can be dropped before reaching their destinations due to congestion, wasting all the resources consumed by them. It is known that, in an extreme case, congestion collapse may happen causing users suffer severe network performance degradation [1]. For congestion control and avoidance, AQM (Active Queue Management) has been investigated. Usually, AQM is implemented in gateways that can distinguish between the propagation delay and persistent queuing delay for effective congestion detection. As a gateway is shared by many active connections with a wide range of round trip times, delay tolerances, and throughput requirements, decisions about the duration and magnitude of transient congestion to be allowed at the gateway are best made by the gateway itself.

The notion of feedback control has been applied to manage the queue length. Most existing work on AQM can be categorized in two major classes: (1) ad hoc event-driven approaches and (2) time-driven feedback control approaches. Ad-hoc approaches are event-driven; however, they neither have a mathematical model nor apply formal feedback control techniques. Other approaches rely on a mathematical model of the TCP and queue dynamics to perform congestion control. They use linear PID (Proportional, Integral and Derivative) and nonlinear fuzzy control techniques based on fixed-interval sampling and control algorithm execution for congestion control. Unlike these approaches, our work is not only event-driven but also based on formal control theoretic techniques. We discuss a few representative existing approaches from the two camps that are closely related to our work in the following.

### *1.1. Ad Hoc Event-Driven Approaches*

RED (Random Early Detection) [2], which is one of the earliest approaches for AQM, controls the queue length based on ad hoc feedback control. The objective of RED is to detect any incipient congestion early and provide congestion notification to the sources to let them reduce their transmission rates before packets are dropped due to overflow in network queues. To detect congestion, RED keeps track of an exponentially weighted moving average of the queue length. If the average queue length exceeds a minimum threshold, RED randomly drops or marks packets with an explicit congestion notification bit. Additionally, all packets are marked or dropped if the average queue length exceeds the specified maximum threshold. RED is an ad hoc feedback-based approach, as it neither mathematically models the TCP and queue length dynamics nor develops a feedback controller for congestion control by applying formal control theoretic techniques [3]. Instead, RED tries to avoid congestion by monitoring the average queue length and manipulating the packet drop rate if necessary. Due to its superior performance to the previous approaches such as the Drop Tail mechanism, it is recommended by the Internet Engineering Task Force and adopted by many commercial routers. However, it is difficult to analyze the dynamics of RED, because the RED mechanism [2] lacks a mathematical model. Another drawback of RED is the difficulty of tuning its parameters. RED received a remarkable attention in the research community. It is followed by a number of projects including [4, 5, 6, 7, 8, 9, 10] just to name a few. Lin et al. proposed Flow Random Early Drop (FRED) [10]. They evaluate the effectiveness of RED over traffic types categorized as non-adaptive, fragile and robust, according to their responses to congestion. They point out that RED allows unfair bandwidth sharing when a mixture of the three traffic types shares a link. FRED uses per-active-flow accounting to impose on each flow a loss rate that depends on the flow's buffer use. Ott et al. proposed Stabilized RED (SRED) [9] to statistically estimate the number of active flows at a link and also identify misbehaving flows. SRED uses the estimated number of active flows and the instantaneous queue size to calculate the packet-dropping probability. Feng et al. developed a self-configuring version of RED [8]. The authors suggest using an on-line algorithm for dynamically changing RED parameters according to the observed traffic. The authors show that this mechanism can reduce packet losses, while maintaining high link utilization. Feng et al. proposed BLUE [7] as a congestion control algorithm to be deployed in gateways. By considering packet loss and link idle events to manage congestion, BLUE significantly outperformed RED.

## *1.2. Time-Driven Feedback Control Approaches*

On the other hand, there have been efforts [11, 12, 13, 14] to apply feedback control theory [3, 15] to AQM, since feedback control is very effective to support the desired performance when the system model includes uncertainties [3, 15]. Misra et al. modeled the interactions of TCP flows and AQM routers by using stochastic differential equations [16]. Using this model in their simulations, they analyzed the impact of RED parameters on the network performance. This mathematical model constitutes a basis for a number of projects regarding AQM. The authors of [17], [11], [12], [13] and [14] used this model to develop control-theoretic approaches for AQM. Hollot et al. analyzed RED in control theoretic aspects [18] using the mathematical model developed in [16]. Using this model, they also developed a P and a PI controller in a companion paper [17]. Zhang et al. proposed an online self-tuning structure [11] based on the mathematical model [16] to estimate and correct network parameters online. Accompanied by a PI controller, this structure is used to manage the queue length. Heying et al. [12] developed a novel algorithm, called Proportional Integral based series compensation and Position feedback compensation (PIP), to manage the queue length. They used the model developed in [16] to implement their approach. Wang et al. [13] proposed an optimized version of the mathematical model in [16] and they built the Adaptive Optimized Proportional Controller (AOPC). AOPC measures the latest packet loss ratio and uses it as a complement to the queue length in order to dynamically adjust the packet drop probability. It measures an additional state information, i.e., the latest packet loss ratio, to enhance the control performance. Fengyuan et al. [14] built a fuzzy logic controller based on the model developed by Misra et al. [16] to manage the queue length. Their approach is inspired by the fact that fuzzy control theory [15] is more versatile than the classical control theoretical solutions in the presence of nonlinear system behaviors. However, their approach is also time-driven.

Although the time-driven feedback controller [17] is shown to achieve better performance than RED, it has shortcomings too. A time-driven controller uses equidistant sampling of the controlled system behavior, e.g., the queue length in a router, in time and compares the measured value to the specified reference to compute the error, e.g., the difference between the current queue length and the specified reference value. Based on the error, the control signal is computed to achieve the desired performance such as the reference queue length. However, a time-driven approach for feedback-based congestion control may not be able to support good performance, if a low sampling rate for feedback control is used or a large number of packets arrive in a short time period that is shorter than the sampling period. To avoid this problem, a short sampling period should be selected based on pessimistic assumptions about the network load. As a result, the controller is executed unnecessarily often when the load is not high, wasting precious resources at the gateway.

### *1.3. Event-Driven AQM based on Formal Control Theory*

In this paper, to seamlessly integrate the event-driven nature of RED and control theoretic approaches for congestion control (and avoidance), we develop an *event-driven feedback controller based on formal control theory* [19, 3]. The key idea of our approach is to design a feedback-based congestion controller that is invoked upon the arrivals of a specified number of packets rather than being invoked at every fixed sampling period. The advantages of our event-driven approach for congestion control are as follows:

- The nature of congestion control is event-driven and initiated by packet arrivals. For this reason, RED is designed to be event-driven. Our approach is also event-driven. Further, it is based on a rigorous mathematical model and formal control theory [19, 3] unlike RED. Thus, we can apply well established control theory [19, 3] to tune and mathematically analyze and support the stability of our feedback-based congestion control scheme.

- If a large number of packets arrive in a short time interval, event-driven controller autonomously executes more often. As a result, the latency for congestion control reduces, enhancing the reactiveness to bursty network loads. In contrast, a time-driven controller has to wait until the next sampling period even in the presence of a dramatic increase in packet arrivals during the current sampling period.

- If the packet arrival rate decreases, an event-driven controller automatically adapts itself to execute less frequently. As a result, under light loads, it consumes less computational resources than a time-driven controller does.

To support event-driven congestion control, we convert the time domain TCP and queue model [16, 18] to the corresponding spatial domain−event domain−model. A summary of our key contributions follows:

- We transform the time-domain nonlinear TCP and queue model [18] to the corresponding spatial-domain model. For this transformation and event-driven controller design, we adapt the event-driven control theoretic techniques [19] developed for motor synchronization. The key idea behind [19] is to measure the time between angular movements around a motor's axis to compute the speed and acceleration in an event-driven fashion rather than calculating the speed and acceleration using a fixed (periodic) sampling rate. In this paper, we adapt this approach to measure the time taken for a specified number of packets to arrive at the queue. Thus, our approach is not tied to a fixed sampling rate but purely driven by events, i.e., packet arrivals.

- We linearize the spatial-domain nonlinear model and design an event-driven controller based on the linearized model. The basic approach is similar to [17] that linearize the time-driven model; however, we linearize the event-driven model in the spatial domain unlike [17].

- We thoroughly evaluate the performance of our approach via an extensive simulation study in OMNeT++ [20]. OMNeT++ is widely used for network research, because it is relatively easy to use due to its modular, component-based, and open-architecture. Further, it provides similar capabilities to the other network simulators such as ns-2 or ns-3. We compare it to five advanced approaches for AQM: (1) RED [2] with the 'gentle' parameter turned on, (2) the time-driven feedback-based PI (Proportional Integral) congestion controller developed by Hollot et al.[17], (3) Proportional Integral based series compensation, and Position feedback compensation (PIP) Controller [12], (4) Adaptive Optimized Proportional Controller (AOPC) [13] and (5) Fuzzy Logic Controller (FLC) [14]. The simulation results show that our event-driven controller effectively maintains the queue length around the specified reference, while reducing queue length fluctuations compared to the tested baseline approaches. At the same time, it achieves shorter E2E (end-to-end) delays and noticeably smaller E2E delay fluctuations than RED, PI and PIP controllers, while achieving almost the same E2E delays and E2E delay fluctuations with AOPC and FLC controllers. Further, it is invoked only 8 times per second in average. In contrast, RED is activated 30 times/s in average while PI [17], PIP [12], AOPC [13] and FLC [14] congestion controllers are activated 160 times/s.

- Moreover, we have repeated the same set of experiments for a higher bandwidth and arrival rate. More specifically, we increase the bandwidth and arrival rate by an order of magnitude for experimental purposes. Interestingly, the time-driven approaches for AQM, i.e., the PI controller, PIP controller, FLC controller and AOPC, which show good performance for a relatively low bandwidth and arrival rate in our simulation study, show poor performance in this set of experiments, because their activation frequency is fixed and do not increase/decrease based on the traffic pattern. RED and our event-driven controller significantly outperform the other approaches in terms of the queue length, E2E delay, and packet drop rate. However, RED is executed more than 2000 times/s in an extreme case. In contrast, our event-driven controller is activated no more than 200 times/s. Also, its average activation frequency is 90 times/s. Our approach is executed less frequently, because it effectively avoids building a large backlog, while the sources adjust their TCP congestion windows accordingly.

The remainder of this paper is organized as follows. A problem formulation is given in Section 2. Our event-driven approach for active queue management is described in Section 3. The performance of our approach and five baselines is compared in Section 4. Finally, Section 5 concludes the paper and discusses future work.

## 2. Problem Formulation

This section describes the scope of this paper, while reviewing the main objectives of AQM.

AQM aims to support congestion avoidance by controlling the queue length to be shorter than the specified upper bound, even if the transport layer protocols in the traffic sources do not support any congestion control. In this way, AQM aims to reduce the number of packet drops when a network traffic burst arrives. Although there is an upper bound on the queue length, AQM is desired to allow queue length fluctuations to let the queue absorb bursty traffic spikes and accommodate transient congestion. This can be accomplished by either controlling the average queue length instead of the transient queue length or by employing a low pass filter, such as the integrator in a PID (proportional, integral, and differential) or PI controller [3], on the control path. Thus, the queue length limit reflects the size of bursts needed to be absorbed rather than the steady state queue length desired to be maintained. Queue length control affects the overall throughput and E2E delay. Especially, there is a trade-off between throughput and E2E delay. Enforcing a smaller upper bound on the queue length translates into lower E2E delay, but this comes at the expense of lower throughput and vice versa.

In AQM, the gateway needs to randomly select a victim connection to drop packets from. If the gateway drops packets from all connections at the same time, this will result in *global synchronization* where all the flows throttle back their transmission rates under congestion and potentially increase the rates later in a synchronized manner. As a result, the network performance may oscillate widely. Via random selection, AQM also aims to avoid *biases against bursty traffic* that is observed, for example, in a Drop Tail queue [2].

Moreover, AQM should be compatible with TCP. Generally, AQM notifies the traffic source of a congestion condition implicitly (by dropping packets) or explicitly (by forwarding an explicit control notification to sources). Therefore, AQM is appropriate to be used with TCP flows that can react to congestion detection.

In addition, there are several features desired to be supported by AQM:

- An AQM algorithm is desired to react quickly to workload changes. Ideally, an overshoot, i.e., the queue length longer than the desired reference value, and settling time, i.e., the time taken to cancel an overshoot (if any), should be minimal in the presence of dynamic load changes.

- Even if the load changes abruptly, the algorithm is desired to control the queue length to be stable and avoid large queue length oscillations.

- The implementation and tuning of the algorithm is desired to be easy.

- It is desirable for the algorithm to not need any further tuning upon load changes.

- The algorithm itself should not consume excessive amounts of system resources.

The bursty nature of network traffic and dynamic behavior of network components make these performance criteria difficult to meet. RED controls the packet drop rate by monitoring the average queue length. However, RED lacks a mathematical or control theoretic model. Thus, it shows weak performance in the presence of highly dynamic network traffic [18]. Also, RED has many parameters to tune, which makes it difficult to obtain best performance. The time driven controllers such as ([17], [12], [13], [14]) showed superior performance to RED; however, they cannot adapt their sampling rate according to the network traffic. Thus, they must be designed in a pessimistic way to support acceptable performance under heavy loads. As a result, they have to sample the queue length and run the AQM algorithm unnecessarily often under light loads as discussed before.

To address these problems, we develop an EDC (Event Driven Controller). Different from most of existing approaches for AQM that is based on either ad hoc event-driven feedback control [2, 4, 5, 6, 9] or time-driven control theoretic techniques [17, 11, 12, 13, 14], our approach for congestion control leverages both the event-driven nature of the RED scheme and control theoretic aspect of the feedback PI controller scheme. EDC has only two parameters to tune for application dependent performance requirements: (1) the desired queue length set-point and (2) event threshold that specifies the number of packet arrivals that triggers an event. By applying control theoretic techniques [19, 3], we aim to tune EDC to support its stability. Also, EDC aims to reduce the consumption of system resources for AQM via systematic event-driven control of the queue length.

## 3. Event-Driven Control for AQM

In this section, the architecture of our event-driven AQM scheme is described. For event-driven AQM, we first transform the time-domain non-linear TCP and queue model [18] that shows the relation between packet arrivals and queue length variations [18] to the spatial-domain correspondent. Also, we linearize the model around the operating point, at which the spatial-domain derivatives of the queue length and TCP window size are zero, in order to support the stability of our EDC. Using the linearized model, an event-driven feedback controller for congestion control is designed as discussed in the following subsections.

### *3.1. System Architecture*

Figure 1 shows the system architecture of our closed loop control system. Arriving packets are added to the queue or randomly dropped by the

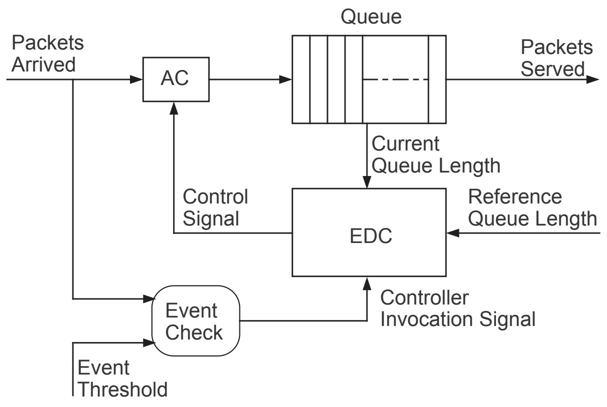

Figure 1: System Architecture

Admission Controller (AC) according to the drop probability calculated by our EDC. EDC is invoked if the number of arriving packets becomes equal to (or exceeds) the specified event threshold. When invoked, EDC computes the difference between the current queue length and specified reference queue length (i.e., set-point) to compute the control signal, i.e., the drop rate adjustment, needed to avoid congestion. If many packets arrive in a short time interval, EDC is invoked frequently and vice versa.

The selection of the event threshold is based on the trade-off between control performance and resource usage: A smaller event threshold value generally makes our EDC more reactive to dynamic loads and vice versa. However, simply picking the smallest possible event threshold may not be a solution, because too frequent EDC invocations may consume excessive computational resources at the router. In our approach, the event threshold is computed based on the network parameters usually available, such as the link capacity, typical packet size, or the set-point queue length. For example, let us assume that the link capacity is 10Mbps and the measured average packet size is 1,000 bits. In this case, to invoke EDC for a number of new packet arrivals estimated to consume 2% link utilization, the event threshold is set to 200 packets. Alternatively, we can select the event threshold considering potential queue length variations. Since the event-driven controller dynamically adapts to the arrival rate, the threshold value is not dependent on the arrival rate, but it is determined based on the tolerable queue length variation that can be specified by a network administrator. For example, when the set-point queue length is 200, the event-driven controller can be activated upon 30 packet arrivals. In Section 4, we use the second approach since selecting the event threshold based on the queue length variation is independent of the network capacity or packet length.

From these examples, we observe that, to select the event threshold, we do not need to assume heavy network loads that may happen only occasionally. Selecting a smaller value as the event threshold or selecting a short sampling period both cause higher consumption of system resources. However, after a sampling period is selected for a high arrival rate that can be predicted, for example, based on historic data arrival rates, the controller is executed at every fixed sampling period, even in the presence of a large increase or decrease of the arrival rate, which could be difficult to predict. This problem is less severe in event-driven control, since the event-driven controller is automatically activated more/less often as the arrival rate increases/decreases. Hence, selecting the event threshold is much less complex and less pessimistic than choosing the sampling period for a time-driven controller.

### *3.2. TCP and Queue Mathematical Models*

Let $\dot{x}$ denote the time derivative of $x$. At time t, the following nonlinear differential equations model the TCP and queue dynamics [18] in the time domain:

$$\begin{aligned}\dot{W}(t) &= \frac{1}{R(t)} - \frac{W(t)\cdot W(t-R(t))}{2R(t-R(t))}\cdot p(t-R(t))\\ \dot{q}(t) &= \frac{W(t)}{R(t)}\cdot N(t) - C\end{aligned} \tag{1}$$

where W is the expected TCP window size, $q$ is the expected queue length, C is the link capacity (packets/s), $N$ is the load factor expressed in terms of the number of TCP sessions, $T_p$ is the propagation delay, R is the average round trip time $= \frac{q}{C} + T_p$ (seconds) and $p(\in [0,1])$ is the packet mark/drop probability.

In Eq. 1, the queue length $q$ and the window size $W$ are bounded positive quantities; therefore, $q \in [0, \bar{q}]$ and $W \in [0, \bar{W}]$ where $\bar{q}$ and $\bar{W}$ denote the maximum queue size and maximum window size, respectively.

### *3.3. Transformation from Time Domain to Spatial Domain*

For event-driven AQM, we transform the window size and queue length models in Eq. 1 to the spatial domain correspondents by applying the techniques presented in [19].

If $\theta(t)$ denotes the number of packets arrived at the queue at time t, the packet arrival rate at time $t$ is:

$$\frac{d\theta}{dt} = \frac{N(t)}{R(t)} \cdot W(t) \tag{2}$$

The transformation from the time domain ($t$) to the spatial domain ($\theta$) can be performed based on this relationship. *The key idea is to consider $\theta$ no longer as a function of time t, but to let time t be a function of packet arrivals $\theta$.* The notation $t(\theta)$ then denotes the time at which $\theta$ packets arrived at the queue. With this interpretation, $W(\theta)$, $R(\theta)$ and $N(\theta)$ denote the window size, round trip time and number of TCP sessions at the time when $\theta$ packets arrive at the queue. Based on this observation, we transform Eq. 2 to the *event-driven, spatial-domain* as follows:

$$dt = \frac{R(\theta)}{N(\theta) \cdot W(\theta)} \cdot d\theta \tag{3}$$

Let $\tilde{W}(\theta) = \frac{dW}{d\theta}$ and $\tilde{q}(\theta) = \frac{dq}{d\theta}$. Since $\dot{W}(t) = \frac{dW(t)}{dt}$ in the time-driven model (Eq. 1), we can transform this model by substituting $dt$ with Eq. 3 and setting the independent variable to $\theta$:

$$\begin{aligned} f(\theta) = \tilde{W}(\theta) &= \frac{R(\theta)}{N(\theta) \cdot W(\theta)} \cdot \left[\frac{1}{R(\theta)} - \frac{W^2(\theta)}{2R(\theta)} \cdot p(\theta)\right] \\ &= \frac{1}{N(\theta) \cdot W(\theta)} - \frac{W(\theta)}{2N(\theta)} \cdot p(\theta) \end{aligned} \tag{4}$$

$$\begin{aligned} g(\theta) = \tilde{q}(\theta) &= \frac{R(\theta)}{N(\theta) \cdot W(\theta)} \cdot \left[\frac{W(\theta)}{R(\theta)} \cdot N(\theta) - C\right] \\ &= 1 - \frac{C \cdot R(\theta)}{N(\theta) \cdot W(\theta)} = 1 - \frac{C(\frac{q}{C} + T_p)}{N(\theta)W(\theta)} \end{aligned} \tag{5}$$

Note that the round trip time delay in $t - R(t)$'s in Eq.1 is omitted while the time domain model is transformed into its spatial domain correspondent (Eq.4 and Eq.5), because the time between two consecutive events $t(\theta) - t(\theta - 1)$ is at least an order of magnitude larger than $R(t)$ due to our method for selecting the event threshold discussed before. Also, note that the RTT is not ignored in the linearized model shown in Eq. 4 and Eq. 5.

After the transformation, we have obtained another set of nonlinear differential equations. In order to apply linear control theory [3] to these models, we linearize them around an operating point next.

### *3.4. Linearizing the TCP and Queue Models*

In this paper, $(W(\theta), q(\theta))$ in Eq. 4 and Eq. 5 is defined as the state of event-driven AQM. Also, the spatial domain expression of the drop probability $p(\theta)$ is the control input to the AC in Figure 1. Therefore, the operating point $(W_0(\theta), q_0(\theta), p_0(\theta))$ is defined by $\tilde{W}(\theta) = 0$ and $\tilde{q}(\theta) = 0$. From this and Eq. 4 and Eq. 5, the following is derived:

$$W_0^2 p_0 = 2 \text{ and } W_0 = \frac{R_0 C}{N} \Rightarrow p_0 = \frac{2N^2}{(R_0 C)^2} \tag{6}$$

where $R_0 = \frac{q_0}{C} + T_p$. Assuming that $N(t) \equiv N$ and $R(t) \equiv R_0$, we linearize the model around the operating point [21] as follows[1]:

$$\begin{aligned} \delta\tilde{W}(\theta) &= j_{11} \cdot \delta W(\theta) + j_{12} \cdot \delta q(\theta) + j_{13} \cdot \delta p(\theta) \\ \delta\tilde{q}(\theta) &= j_{21} \cdot \delta W(\theta) + j_{22} \cdot \delta q(\theta) + j_{23} \cdot \delta p(\theta) \end{aligned} \tag{7}$$

where $\delta X = X - X_0$ and $j_{ij}$'s are elements of Jacobian Matrix [2] of the system:

$$J = \begin{bmatrix} j_{11} & j_{12} & j_{13} \\ j_{21} & j_{22} & j_{23} \end{bmatrix} = \left. \begin{bmatrix} \frac{\partial f}{\partial W} & \frac{\partial f}{\partial q} & \frac{\partial f}{\partial p} \\ \frac{\partial g}{\partial W} & \frac{\partial g}{\partial q} & \frac{\partial g}{\partial p} \end{bmatrix} \right|_{(f_0, g_0)}$$

which can be calculated by taking partial derivatives of the nonlinear model around the operating point $(f_0, g_0)$ and using relationships given in Eq. 6:

[1]Note that the equilibrium points, i.e., $N, R_0, W_0$, and $p_0$, are not assumed to be constant in our system model (Eq. 9). If the system diverges from the equilibrium points, the control performance may degrade. However, the system will stay stable as analyzed in the last paragraph of Section 3.4

[2]The third row of Jacobian Matrix, which is the feedback controller model, is omitted since it is not needed to linearize the nonlinear open loop model.

$$
\begin{aligned}
j_{11} &= \frac{\partial f}{\partial W} = \frac{-1}{NW_0^2} - \frac{p_0}{2N} = \frac{-p_0}{N} = \frac{-2N}{(R_0C)^2} \\
j_{13} &= \frac{\partial f}{\partial p} = \frac{-W_0}{2N} = \frac{-R_0C}{2N^2} \\
j_{21} &= \frac{\partial g}{\partial W} = \frac{-CR_0}{N} \cdot \frac{-1}{W_0^2} = \frac{N}{R_0C} \\
j_{22} &= \frac{\partial g}{\partial q} = \frac{-1}{NW_0} = \frac{-1}{R_0C} \\
j_{12} &= j_{23} = 0
\end{aligned}
\tag{8}
$$

Finally, we derive our linearized spatial domain model as follows:

$$
\begin{aligned}
\delta\tilde{W}(\theta) &= -\frac{2N}{(R_0C)^2} \cdot \delta W(\theta) - \frac{R_0C}{2N^2} \cdot \delta p(\theta) \\
\delta\tilde{q}(\theta) &= \frac{N}{R_0C} \cdot \delta W(\theta) - \frac{1}{R_0C} \cdot \delta q(\theta)
\end{aligned}
\tag{9}
$$

The eigenvalues of the linearized TCP and queue dynamics (9) are $-\frac{2N}{(R_0C)^2}$ and $-\frac{1}{R_0C}$, respectively. Since all the network parameters are positive quantities, these negative eigenvalues indicate that the equilibrium state of the non-linear dynamics is locally asymptotically stable. A steady state equilibrium is locally asymptotically stable if there exists an $\epsilon$ neighborhood of the steady state equilibrium such that from an arbitrary initial condition within this neighborhood, the system converges to this steady state equilibrium. Formally, a steady state equilibrium, $\bar{y}$, of the difference equation $y_{\theta+1} = ay_\theta + b$ is locally asymptotically stable, if $\lim_{\theta\to\infty} y_\theta = \bar{y}$ $\forall$ initial condition $y_0$ such that $|y_0 - \bar{y}| < \epsilon$ for some small $\epsilon > 0$ [22].

### *3.5. Event-Driven Congestion Controller Design*

For event-driven congestion control, we implement a PI (Proportional and Integral) controller to manage the drop probability for congestion control. A PI controller is a variation of a popular PID (Proportional, Integral and Differential) controller. A proportional controller computes the control signal in proportion to the error, i.e., the difference between the measured performance of the controlled system and the specified reference performance. A proportional controller by itself cannot support the stability of the feedback control system [3]. An integrator is a low pass filter and it can support the stability of the closed-loop system. In this paper, an integrator is employed to allow traffic bursts as discussed before, while supporting the stability of the closed-loop congestion control system. We do not use a differential controller, because it may show unreliable performance when workloads are highly dynamic. The time-driven congestion controller developed by Hollot et al. [17] is also implemented using a PI controller.

We formulate the PI Controller as a transfer function, which characterize the behavior of a closed-loop system [3] in the discrete time $z$ domain rather than in the continuous time $s$ domain, because most of computational systems, such as routers, usually work in the discrete time domain. Further, the notation $\tilde{z}$ is used instead of $z$ in the following PI transfer function to emphasize that the discretization has been made in the spatial domain, instead of the time domain:

$$C(z) = \frac{\alpha(\tilde{z} - \beta)}{\tilde{z} - 1} \tag{10}$$

where $\alpha = K_P(K_I + 1)$, $\beta = \frac{1}{K_I+1}$. $K_P$ and $K_I$ are the proportional and integral control gains. To support the stability of the closed-loop system for AQM, we need to tune the control gains, $K_P$ and $K_I$, so that the closed-loop poles are located inside the unit circle [3]. There are several approaches to tune control gains to achieve the objective. In this paper, we apply the Root Locus method [3], in which one can graphically tune feedback control gains to support the stability by locating the closed-loop poles inside the unit circle. In this way, we ensure that our EDC closely supports the set-point queue length, while avoiding excessive queue length oscillations, i.e., sensitivities of our EDC to bursty packet arrivals, as experimentally verified in Section 4. For more details about controller tuning and the Root Locus method, readers are referred to [3].

## 4. Performance Evaluation

Performing an extensive simulation study similar to [17, 12, 13, 14], we compare the performance of EDC to five existing approaches used as baselines in this paper: (1) RED [2] with the 'gentle' parameter turned on, (2) PI [17], (3) PIP [12], (4) AOPC [13] and (5) FLC [14] congestion controllers. For performance evaluation, we simulate 10 clients and a server connected

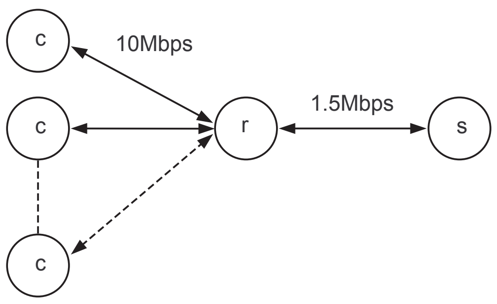

Figure 2: Simulation Setup

by a gateway in INET framework [23] which is an open-source communication networks simulation package for the OMNeT++ simulation environment [20]. The accuracy of the INET simulation package is verified in [24] by using a testbed that consists of INET nodes connected to real nodes through the Ethernet. OMNeT++ is widely used for general network research due to its accuracy and modular, component-based design [25]. As shown in Figure 2, each client is connected to the gateway with a 10 Mbps link. The 1.5Mbps link between the gateway and the server constitutes a bottleneck. Our simulation starts with 60 ftp and 180 http applications and continues until 100s. At 100s, 60 more ftp applications are activated in order to observe how the tested approaches for AQM react to the abrupt load increase. The simulation ends at 200s. In addition, we perform another set of experiments to show the performance of the tested approaches for relatively high bandwidth and high arrival rate. Specifically, we increase the bandwidth of the links between the servers and gateway to 100 Mbps and bandwidth of the link between gateway and client to 15 Mbps. We also increased the amount of data to be transferred in each ftp connection by the factor of 10, while keeping the other parameters the same. For the clarity of presentation, in the rest of the paper, we call the first set and second set of experiments low bandwidth low arrival rate (LL) and high bandwidth high arrival rate (HH) experiments, respectively.

### *4.1. Credibility of the Simulation Study*

Credibility of a simulation study is an issue. Pawlowski et al [26] have made three invaluable recommendations for a credible simulation study:

- Use a valid simulation model.

- Use appropriate pseudo-random number generators (PRNGs).
- Derive the statistical confidence of simulation results.

Our simulation model described before is designed after the models used in the previous congestion control work [17, 12, 13, 14]. We have designed our event-driven controller by using well established formal control theoretic tools [19, 3]. For the tested baseline approaches, we set the congestion control parameters by following the recommendations provided in the original papers for each controller, while tuning the control gains of the PI and PIP controllers to obtain the best possible performance and support their stability by applying formal control theoretic techniques [3]. The control parameters used for our simulation study are given in Section 4.2.

Secondly, we have used the built-in PRNG of OMNet++. OMNeT++ supports modern PRNGs such as Mersenne Twister used in our simulation study. Mersenne Twister provides a super astronomical cycle of $2^{19937}-1$ and good virtual randomness in up to 623 dimensions ([26]). Further, OMNet++ supports automatic pseudo random seeding. Specifically, OMNeT++ selects a seed number from a table of 256 seeds, spaced about 8 million values apart. Thus, our simulation runs use different seed numbers largely spaced apart.

Finally, we repeated each simulation 10 times and took their average as simulation results. We computed 90% confidence interval for each average performance data, i.e., the average queue length and delay. A confidence interval is computed for each data point of the average curve in Figures 3 – 10. However, we omit the confidence interval bars, because their values are very small ranging between 0.5% – 2.7% and unnecessarily clutter the performance result graphs.

### *4.2. Parameter Settings*

The parameters of the baselines and our event driven controller are set as follows:

- **Gentle RED:**
  - Minimum queue length threshold: 150 packets
  - Maximum queue length threshold: 300 packets
  - Averaging weight ($wq$ in [2]): 0.00133
  - Maximum drop probability: 0.1

- **PI Controller**
  - Reference (set-point) queue length: 200 packets
  - Sampling frequency: 160 Hz (as recommended in [17])
  - $K_P = 0.0015$ and $K_I = 0.0003$ (tuned via the Root Locus design method [3])
- **PIP**
  - Reference queue length: 200 packets
  - Sampling frequency: 160 Hz
  - $K_h = 0.0021, \tau = 0.47$ (tuned via the Root Locus design method [3])
- **AOPC**
  - Reference queue length: 200 packets
  - Sampling frequency: 160 Hz
- **FLC**
  - Reference queue length: 200 packets
  - Sampling frequency: 160 Hz
  - $K_e = 0.05$ and $K_{\delta e} = 0.01$ (tuned via trial and error)
- **EDC:**
  - Reference queue length: 200 packets
  - Event threshold: 30 packets
  - $K_P = 0.03$ and $K_I = 0.01$ (tuned via the Root Locus design method [3])

As discussed in Section 3.1 the selection of the event threshold is based on the trade-off between control performance and resource usage. Since the event-driven controller dynamically adapts to the arrival rate, the threshold value is not dependent on the arrival rate, but it is determined based on the tolerable queue length variation as discussed before. In this paper, the maximum buffer size is 800 packets for all the tested approaches. We picked a threshold value of 30 packets that can support reasonable performance in terms of queue length variation, i.e., 3.75% of the maximum buffer size or 15% of the reference queue length for EDC.

## *4.3. Performance Evaluation Results*

For performance analysis, we show (1) the average and transient queue length, (2) E2E delay, (3) number of packet drops and (4) controller activation frequencies. Since the measured link utilization is almost 100% for all the tested approaches, we do not plot it.

Figure 3 shows the transient and average queue lengths of all the baselines and our controller, EDC. From the figure, we observe that RED keeps the average queue length between the specified minimum and maximum thresholds, i.e. 150 and 300 packets, after canceling big transient overshoots (in terms of the queue length) at the beginning. However, RED's reaction to the abrupt load increase at 100s is much slower than the other controllers' reactions. All the other approaches successfully maintain the average queue length near the desired set-point of 200 packets, converging to the set-point. When the load is increased suddenly at 100s, every approach shows a queue length overshoot. PI, PIP, AOPC and FLC show relatively large transient fluctuations of the queue length around the set-point. This is mainly because of their fixed activation rate, which may not be high enough to effectively handle an abrupt increase of the packet arrival rate. Generally, a short sampling period (i.e., a high sampling rate) usually improves the performance of the feedback controller, but it increases the control overhead [3]. Furthermore, simply using a short sampling period for feedback control may lead to the design of an overly reactive controller, which may show unstable behaviors. In general, selecting an optimal sampling period is a hard problem [3]. The problem becomes even harder for a network congestion control system that has to deal with stochastic workloads unlike physical systems, e.g., a cruise control system, which can model the controlled system, e.g., an automobile, using physics laws. In contrast, EDC does not use a fixed activation rate for congestion control. Instead, it is activated in an event-driven manner; therefore, its activation rate is automatically adapted according to potential variations in the packet arrival rate. As a result, EDC in Figure 3 can considerably reduce the queue length fluctuations compared to the baseline approaches.

Figure 4 shows the transient and average queue lengths of all the baselines and our controller, EDC, under the high bandwidth and the high arrival rate.

As shown in the figure, RED keeps the queue length between 150 and 300 packets. EDC shows the best performance; that is, it closely supports the set-point queue length. In comparison, AOPC, PI, PIP and FLC that are time-driven cannot adapt to the new arrival rate and fail to support the set-point queue length. Since the time-driven controllers, i.e., AOPC, PI, PIP and FLC, are tuned and their sampling rate is selected according to the low traffic rate, these controllers cannot react to the high packet rate which is ten times higher than the initial rate. As a result, they cannot avoid queue length saturation at the maximum buffer size, which is 800 packets for all the tested approaches. To support congestion control for a significantly higher traffic rate than the rate used for tuning these controllers, their sampling rate needs to be higher and their control gains, such as $K_P$ and $K_I$, need to be retuned. Although it is possible to retune control gains and select a new sampling period considering new network conditions, the queue length and delay may largely fluctuate while the control gains and sampling period are adjusted. After the adjustment is finished, the traffic condition may change again. In contrast, EDC does not need to re-tune its parameters, since its sampling rate automatically adapts to higher traffic rate.

Figure 5 shows the average and transient E2E delays for all the baselines and EDC. In Figure 5, the average E2E delay of RED is approximately 0.75s, whereas all the other baselines and EDC maintain the average E2E delay around 0.5s. Further, from the transient E2E delay curves in Figure 5, we observe that EDC achieves smaller E2E delay fluctuations than RED, PI and PIP controllers, while achieving almost equal E2E delay fluctuations as AOPC and FLC. (Note that all the E2E delays plotted in Figure 5 are larger than zero. Relatively small E2E delays shown in the figure are tens of milliseconds.)

Figure 6 shows the average and transient E2E delays for all the baselines and EDC under the high bandwidth and the high arrival rate. In comparison to the Figure5, RED and EDC reduces the E2E delay by effectively taking advantage of the network bandwidth increased by a factor of 10. On the other hand, AOPC, PI, PIP and FLC cause an increase in the E2E delay, since they fail to manage the queue length as previously shown in Figure 4.

Figure 7 shows the average and transient packet drop rates measured in terms of the number of packet drops/s for all the baselines and EDC. In average, RED drops approximately 30 packets/s, whereas all the other baselines and EDC drop approximately 50 packets/s. RED drops a smaller number of packets, since its queue length is generally longer than the reference queue length of other baselines and EDC as shown in Figure 3.

Figure 8 shows the average and transient packet drop rates measured in terms of the number of packet drops per second for all the baselines and EDC under the high bandwidth and the high arrival rate. In comparison to Figure 7, the drop rate increases for all the tested approaches, since more packets are needed to be dropped by the controllers in order to keep the queue length near the set-point. However, AOPC, PI, PIP and FLC cause considerably more drops than RED and EDC do, since they cannot effectively manage the queue length as discussed before.

Finally, we compare the performance of the tested AQM schemes in terms of the activation frequency, i.e., the number of control algorithm executions per second. Figure 9 shows the activation frequencies of RED and EDC. PI, PIP, AOPC and FLC's activation frequency is fixed at 160Hz following [17]. From the figure, we observe that EDC has the smallest transient activation frequencies among the tested approaches. In average, EDC is activated only approximately 8 times per second and RED is activated approximately 30 times/s. As shown in Figure 9, RED and EDC increase their activation frequencies upon the abrupt load increase at 100s and then decrease the frequencies as the load becomes stable after 100s.

Figure 10 shows the activation frequencies of RED and EDC in the HH experiments. PI, PIP, AOPC and FLC's activation frequency is 160 Hz. Compared to Figure 9, the activation frequency of RED and EDC has been increased to adapt to the new arrival rate. Specifically, the average activation frequency of RED has increased from 30 times/s to 350 times/s, exceeding the activation frequency of PI, PIP, AOPC and FLC. In contrast, EDC's average activation frequency has only increased to 90 times/s. The reasonable increase in the activation frequency of RED and EDC is caused by the congestion control mechanism supported by TCP itself, which adapts the transmission rate according to the packet drop rate. If the gateway drops packets, TCP reduces the window size; therefore, the packet arrival rate at the gateway reduces. Additionally, although the average activation frequency of EDC is smaller than the activation frequency of the time driven approaches, EDC based on formal control theoretic techniques effectively adapts its activation frequency according to the arrival rate. Sometimes the *transient* activation frequency of EDC is higher than that of the time-driven approaches. At other times, it is activated less frequently than the time-driven approaches based on the need for activation. As a result, it can effectively manage the queue length, while keeping its *average* activation frequency and resource usage relatively low compared to the time-driven approaches and RED.

## 5. Conclusions and Future Work

Active Queue Management (AQM) is investigated to avoid incipient congestion in gateways to complement congestion control run by the transport layer protocol such as the TCP. To seamlessly integrate the advantages of both event-driven and control-theoretic time-driven approaches, we present an event-driven feedback control approach for AQM based on formal control theory. As our approach is based on a mathematical model, its performance is more predictable than ad hoc event-driven approaches are. Also, it is more reactive to dynamic load changes than time-driven approaches due to its even-driven nature. We analyze the stability of the open-loop model and tune the event-driven controller to support the stability of the closed loop system by applying formal control theoretic techniques. Our simulation results show that our event-driven controller effectively maintains the queue length around the specified set-point. It generally achieves shorter E2E (end-to-end) delays and smaller E2E delay fluctuations than the baselines. Further, our AQM algorithm is invoked much less frequently than the tested baselines. In the future, we will continue to investigate more cost-effective approaches for active queue management.

## References

[1] V. Jacobson, Congestion avoidance and control, in: ACM SIGCOMM, August, 1988, Stanford, CA, USA, pp. 314–329.

[2] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (1993) 397–413.

[3] C. Phillips, H. Nagle, Digital control system analysis and design, Prentice Hall, 1995.

[4] S. Floyd, Recommendations on using the gentle variant of RED, 2000. Available at http://www.icir.org/floyd/red/gentle.html.

[5] S. Floyd, R. Gummadi, S. Shenker, Adaptive RED: An algorithm for increasing the robustness of RED's active queue management, 2001. Available at http://www.icir.org/floyd/papers/adaptiveRed.pdf.

[6] J. Sun, K.-T. Ko, G. Chen, S. Chan, M. Zukerman, PD-RED: To improve the performance of RED, IEEE Communications Letters 7 (2003) 406 – 408.

[7] W. Feng, D. Kandlur, D. Saha, K. Shin, The Blue active queue management algorithms, IEEE/ACM Transactions on Networking 10 (2002) 513–528.

[8] W. Feng, D. D. Kandlur, D. Saha, K. G. Shin, A self-configuring RED gateway, in: IEEE INFOCOM, March, 1999, New York, NY, USA, pp. 1320–1328.

[9] T. J. Ott, T. V. Lakshman, L. H.Wong, SRED: Stabilized RED, in: IEEE INFOCOM, March, 1999, New York, NY, USA, pp. 1346–1355.

[10] D. Lin, R. Morris, Dynamics of random early detection, in: ACM SIGCOMM, September, 1997, Cannes, French Riviera, FRANCE, pp. 127–137.

[11] H. Zhang, D. Towsley, C.V.Hollot, V. Misra, A self-tuning structure for adaptation in TCP/AQM networks, in: ACM SIGMETRICS, June, 2003, San Diego, CA, USA, pp. 302–303.

[12] Z. Heying, L. Baohong, D. Wenhua, Design of a robust active queue management algorithm based on feedback compensation, in: ACM SIGCOMM, August, 2003, Karlsruhe GERMANY, pp. 277–285.

[13] J. Wang, L. Rong, Y. Liu, A robust proportional controller for AQM based on optimized 2nd order system model, Computer Communication 1 (2008) 2468–2477.

[14] R. Fengyuan, R. Yong, S. Xiuming, Design of a fuzzy controller for active queue management, Computer Communication 1 (2002) 874–883.

[15] K. M. Passino, S. Yurkovich, Fuzzy control, Addison Wesley, 1998.

[16] V. Misra, W. B. Gong, D. Towsley, Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED, in: ACM SIGCOMM, August-September, 2000, Stockholm, SWEDEN, pp. 151–160.

[17] C. Hollot, V. Misra, D. Towsley, W. B. Gong, On designing improved controllers for AQM routers supporting TCP flows, in: IEEE INFOCOM, April, 2001, Anchorage, Alaska, USA, pp. 1726–1734.

[18] C. Hollot, V. Misra, D. Towsley, W. B. Gong, A control theoretic analysis of RED, in: IEEE INFOCOM, April, 2001, Anchorage, Alaska, USA, pp. 1510–1519.

[19] W. Heemels, R. Gorter, A. van Zijl, P. van den Bosch, S. Weiland, W. Hendrix, M. Vonder, Asynchronous measurement and control: A case study on motor synchronization, Elsevier Control Engineering Practice 7 (1999) 1467–1482.

[20] OMNeT++, A discrete event simulation environment, http://www.omnetpp.org, 2010.

[21] Z. Vukic, Nonlinear Control Systems, Marcel Dekker, 2003.

[22] O. Galor, Discrete Dynamical Systems, Springer, 2007.

[23] INET, Open-source communication networks simulation package for OMNeT++, http://inet.omnetpp.org/, 2010.

[24] M. Tüxen, I. Rüngeler, E. P. Rathgeb, Interface connecting the INET simulation framework with the real world, in: ICST SIMUTOOLS, March, 2008, Marseille, FRANCE. Article No: 40.

[25] OMNeT++, Publications, http://www.omnetpp.org/publications, 2010.

[26] K. Pawlikowski, H.-D. J. Jeong, J.-S. R. Lee, On credibility of simulation studies of telecommunication networks, IEEE Communications Magazine 40 (2002) 132–139.

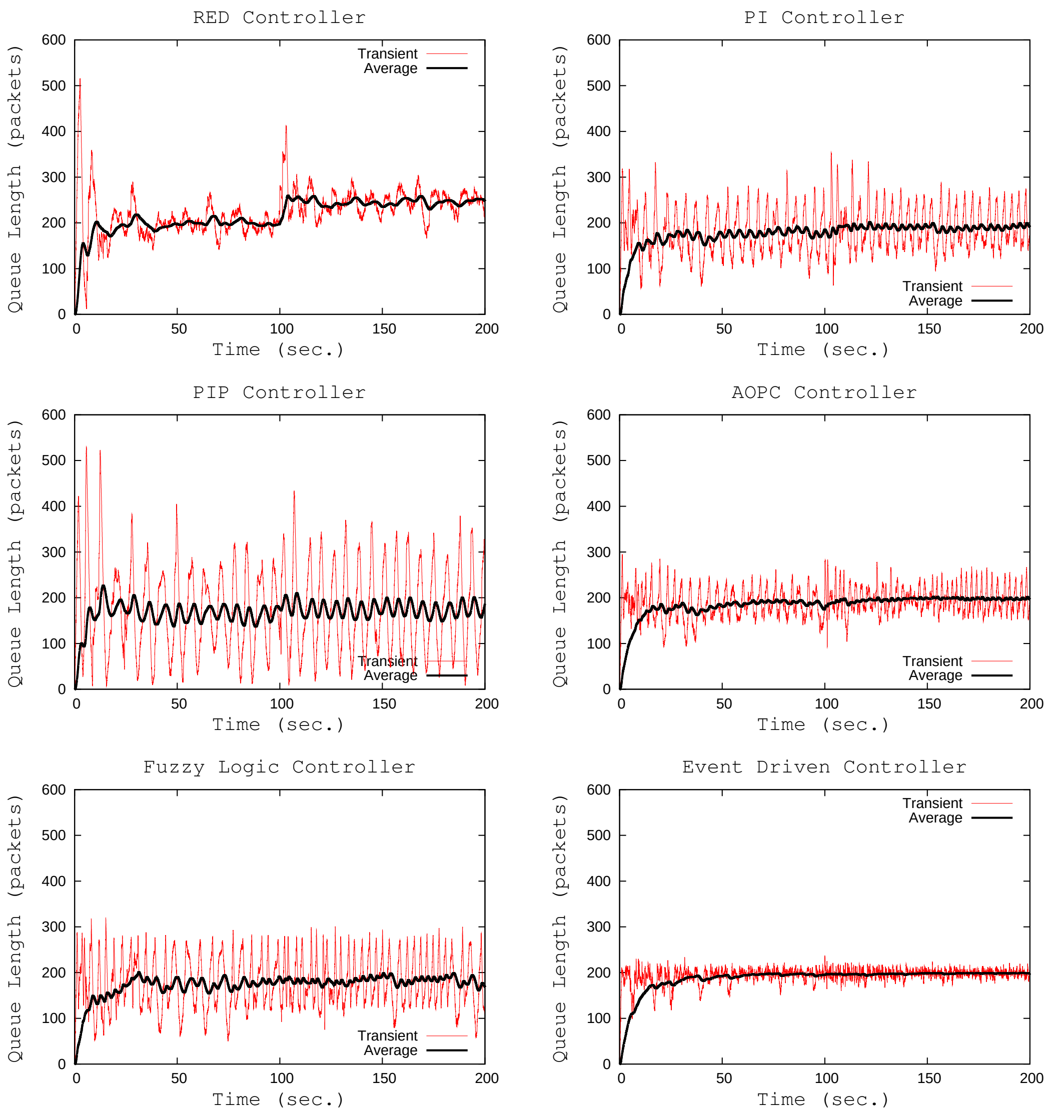

Figure 3: Queue Lengths for LL (Low bandwidth Low arrival rate) Experiments

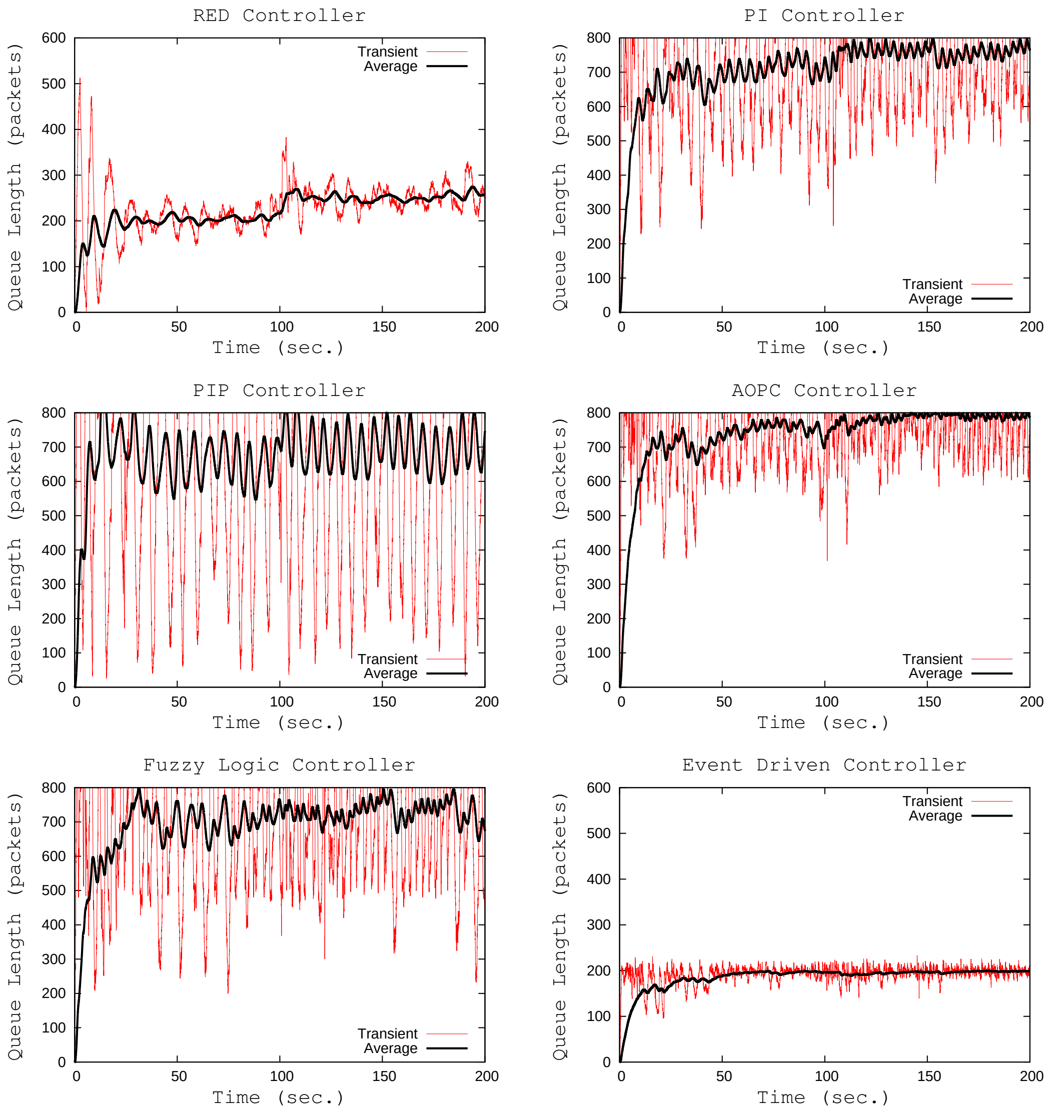

Figure 4: Queue Lengths for HH (High bandwidth High arrival rate) Experiments

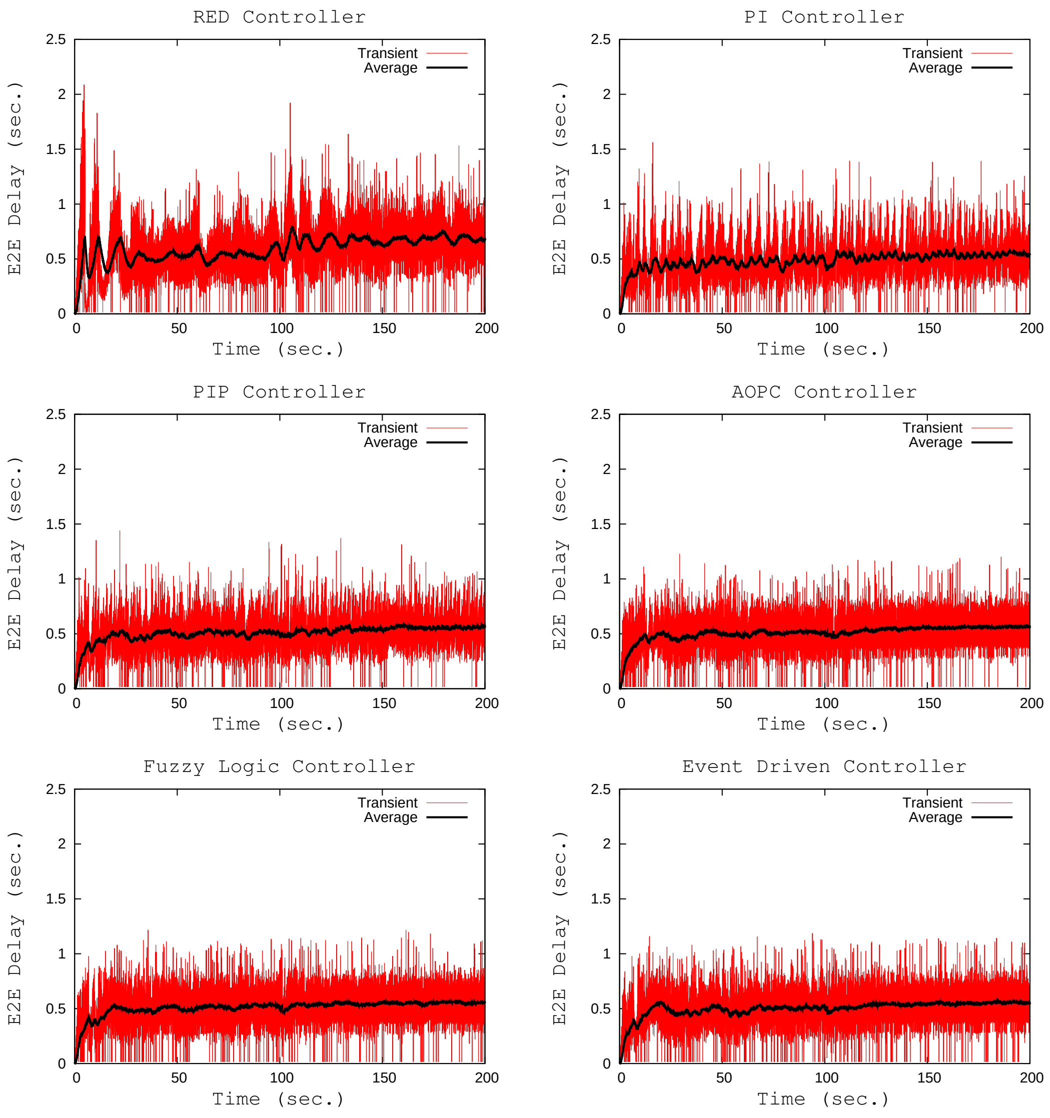

Figure 5: E2E Delays for LL Experiments

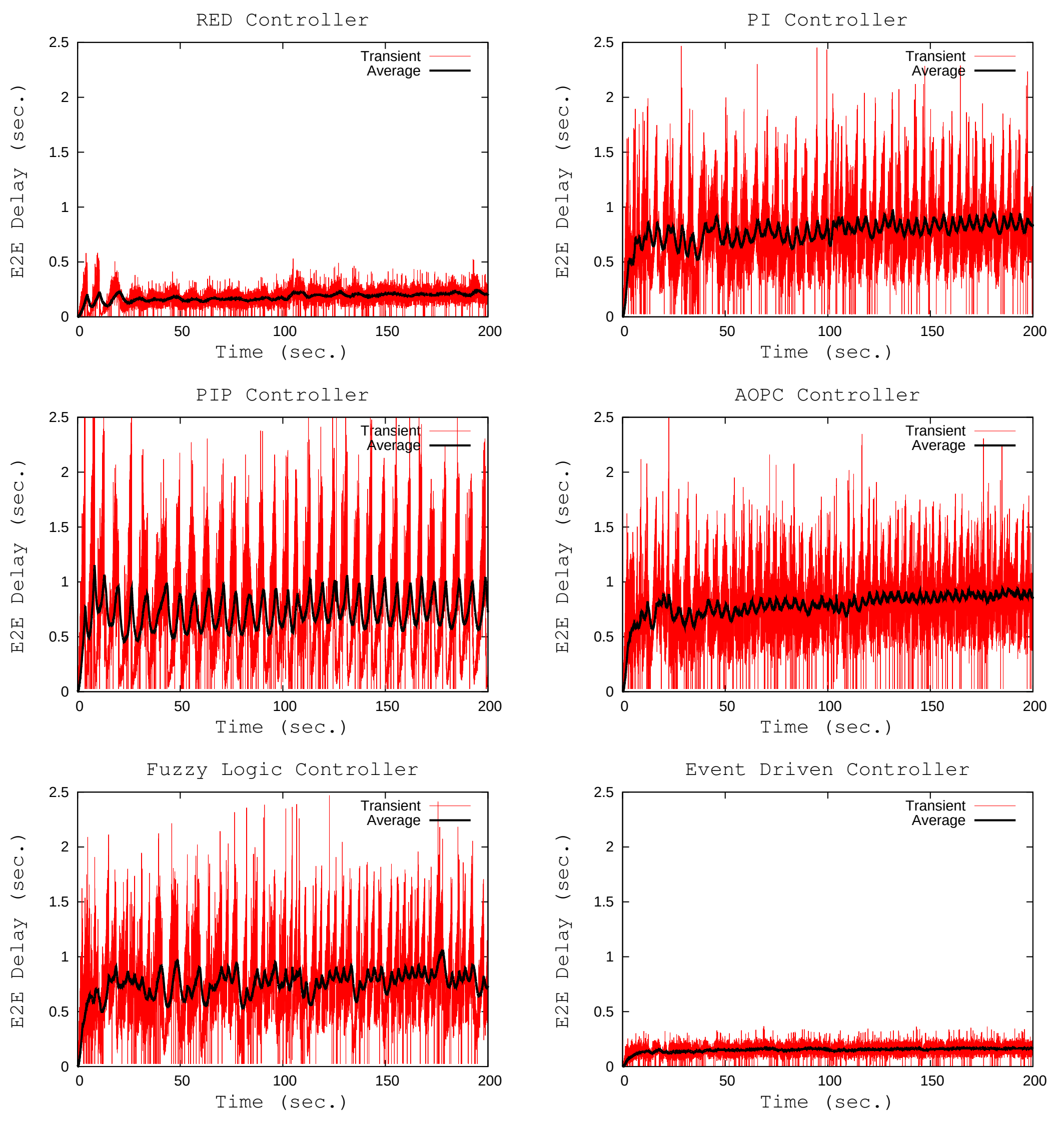

Figure 6: E2E Delays for HH Experiments

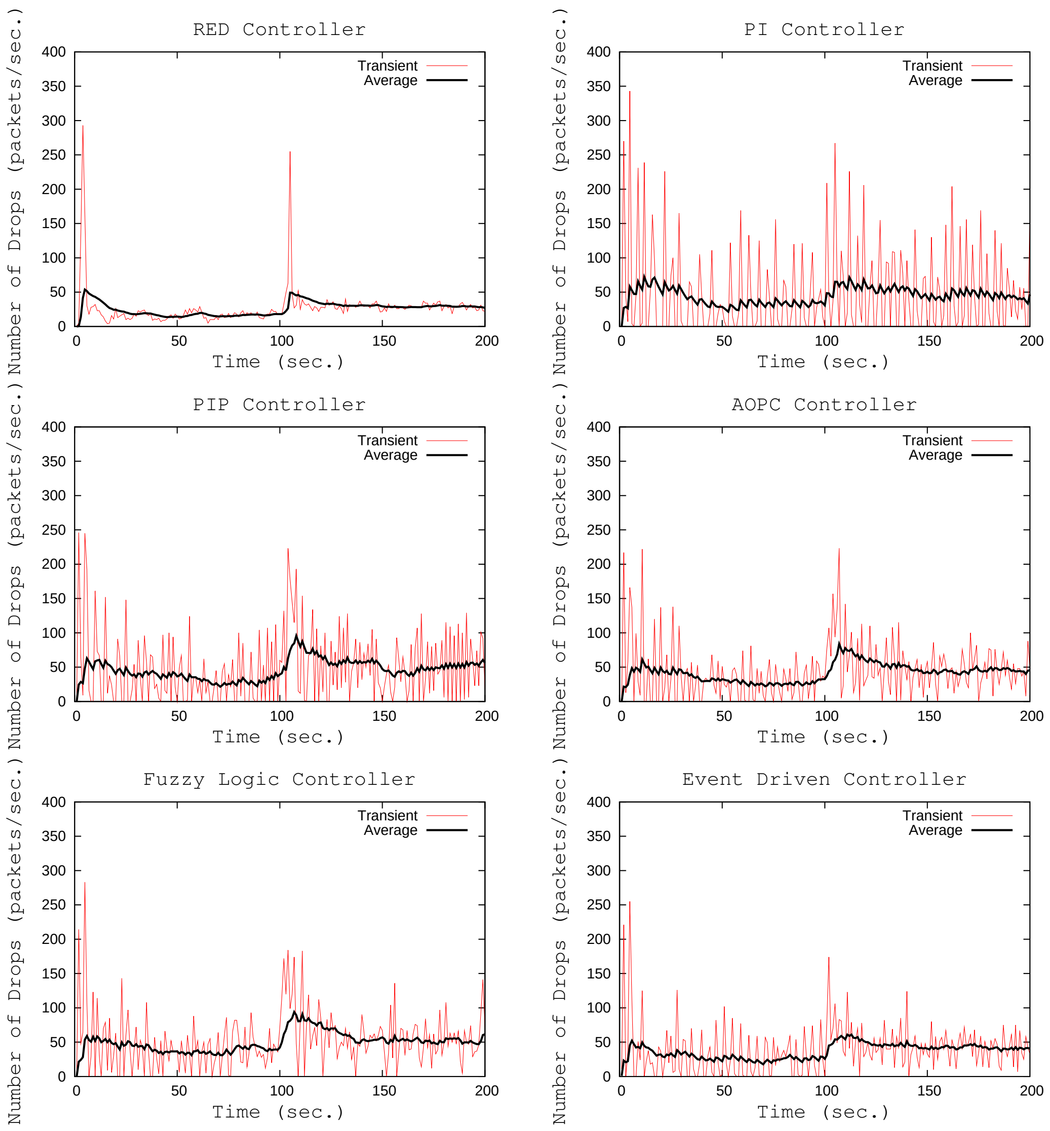

Figure 7: Packet Drop Rates for LL Experiments

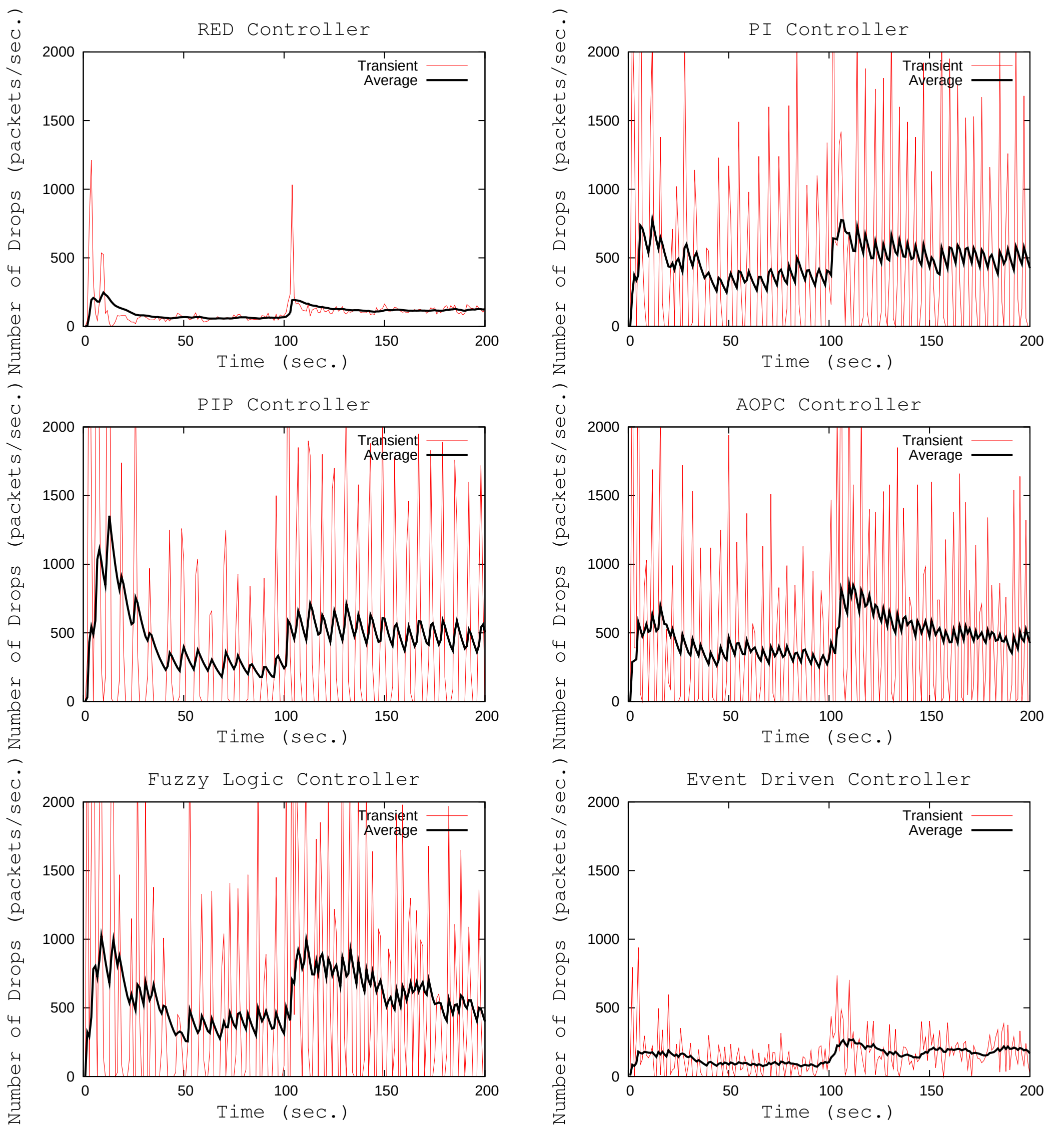

Figure 8: Packet Drop Rates for HH Experiments

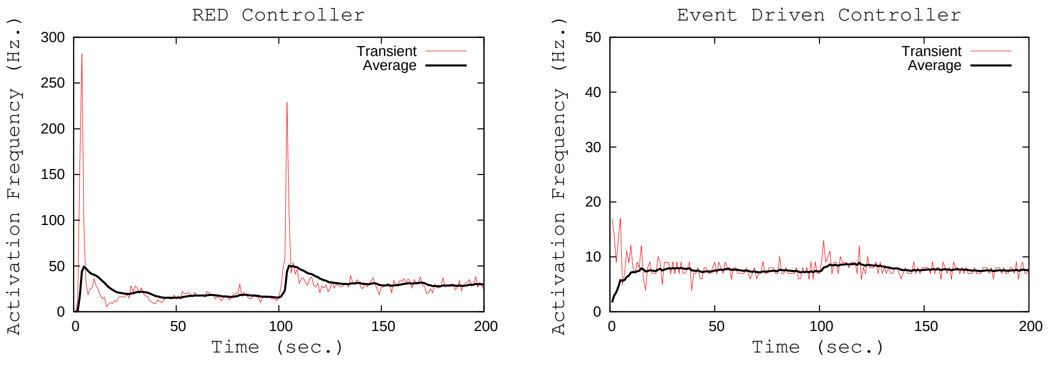

Figure 9: Activation Frequencies for LL Experiments

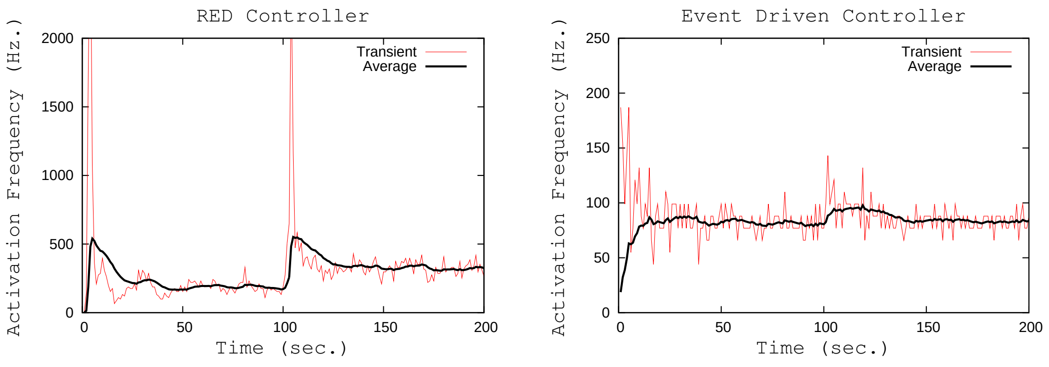

Figure 10: Activation Frequencies for HH Experiments