Aggregate Rate Control for Efficient and Practical Congestion Management

Jae Chung and Mark Claypool
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609, USA
{goos|claypool}@cs.wpi.edu

Abstract— **Active queue management (AQM) promises to overcome the current limitations of end-host only congestion control by providing congestion feedback information before router queue buffers overflow. Many emerging AQM approaches use proportional integral (PI) controller design because of PI's simplicity and effectiveness. Unfortunately, these promising AQMs still face a critical deployment challenge since there are no simple and effective PI control parameter configurations available for time-delayed systems (i.e. the Internet). As a solution, we present the Aggregate Rate Controller (ARC), a reduced parameter PI controller for Internet traffic. ARC, founded on both classical control theory and a sound understanding of Internet congestion control, uses a low frequency rate-based approach to detect congestion that minimizes control noises and provides more flexible link Quality of Service (QoS) compared with queue-based approaches. In addition, we provide practical configuration guidelines for ARC that produce efficient and resilient performance over a wide range of traffic conditions. Simulations verify that ARC effectively handles network congestion over a range of network and traffic conditions, overall outperforming other mechanisms in terms of queue dynamics, link utilization, data loss rate and object response time for Web traffic.**

Keywords: Stochastic processes/Queuing theory, Control theory, Simulations

I. Introduction

TCP, the de-facto Internet transport protocol, has an end-host congestion control mechanism that has largely been effective in managing Internet congestion. Yet, TCP alone can be inefficient in controlling congestion, mainly since end-hosts typically must wait until router buffers overflow before detecting congestion. Active queue management (AQM) with explicit congestion notification (ECN) [1] promises to overcome the limitations of end-host only congestion control by providing congestion feedback information to the end-hosts before router buffers overflow.

The most promising approaches model AQM as a feedback controller on a time-delayed response system and apply control-engineering principles to design efficient controllers for TCP traffic [2], [3], [4]. In modern control systems, proportional integral derivative (PID) designs dominate because of their simplicity and effectiveness. Without exception, this applies to recent AQM developments and has moved AQM research from issues with basic framework design and detailed controller design to, with this paper, issues with practical implementation.

Among the PID principles, AQMs primarily consider only the proportional integral (PI) feedback control since the effect of the derivative control is often insignificant under practical Internet environments. While PI control approaches seems promising, a critical deployment challenge is the configuration of PI control parameters in a time-delayed feedback system (i.e., the Internet); there are no simple and effective PI control parameter configuration available for time-delayed systems [5]. The existing PI control-based AQM mechanisms such as the PI controller [3], Adaptive Virtual Queue (AVQ) [4] and Random Early Marking (REM) [6], lack complete configuration guidelines for network operators making their practical deployment difficult.

In this work, we present a a practical PI control-based AQM offering aggregated rate control (ARC) for TCP traffic. ARC is founded on classic control theory and a sound understanding of PI behavior for the Internet traffic control domain, yielding a reduced parameter PI controller and easy configuration while keeping proven system stability characteristics. We model a TCP-ARC feedback control system using a linear TCP model [3] and develop practical yet effective ARC configuration guidelines. The guidelines cover issues in choosing a target stable boundary system for ARC configuration and provide a method for selecting control parameters that help avoid system instability even when the system is out of the stability boundary. The guidelines also address the effects of the rate sampling interval on system stability, a consideration often neglected in other AQM studies.

Controllers with the same underlying principle design can result in noticeably different implementations, in terms of both complexity and performance, depending on how control information is obtained and feedback is processed. AQMs require information on the incoming traffic load in order to make effective congestion control decisions, information that can be obtained in two ways: by deriving samples of the queue length, or by comparing the incoming traffic rate to the service rate. ARC takes the latter, a rate-based control information acquisition approach. For a small amount of data collection overhead, rate-based data acquisition reduces sampling noise that can significantly degrade the accuracy of congestion measurement. In addition, rate-based mechanisms can more effectively react to impending congestion, making control decisions before outbound queue buildup and can also be tuned to tradeoff queuing delay for link utilization, allowing enhanced support for quality of service (QoS) for various Internet applications.

Through an extensive simulation study, we evaluate ARC and compare it to alternate AQM mechanisms including the PI controller [3], AVQ [4] and SFC [2], and drop-tail queue management for a wide range of network and traffic conditions, including Web flash crowds and multiple congestion bottlenecks. Our simulations validate the stability and practicality of ARC, showing that ARC efficiently handles network congestion in all the tested traffic conditions, and when considering all traffic scenarios, outperforms other mechanisms in terms of queuing delay, link utilization, data loss rate, and Web object response time.

The rest of paper is organized as follows: Section II describes the design of ARC; Section III discusses ARC configuration issues and provides practical guidelines for ARC configuration; Section IV evaluates ARC performance through simulation and compares ARC to several popular AQMs; and Section VI summarizes our conclusions and presents possible future work.

II. Aggregated Rate Controller

Aggregate Rate Controller (ARC) is a rate-based active queue management mechanism for TCP traffic featuring a proportional-integral controller with a minimal set of configuration parameters. ARC provides flexible link QoS and simple yet effective aggregated traffic control based on the TCP congestion control system.

In modern control systems, proportional integral derivative (PID) designs are the most widely used for feedback controllers because of their simplicity and effectiveness, and have recently been applied to active queue management. The PI controller [3] adapts a proportional-integral (PI) controller into a TCP congestion control system using a linear TCP behavioral model, and converts the continuous time-domain PI control function into an algorithmic implementation through discretization. The PI controller can be configured to support a wide range of traffic conditions. However, it is still not mature enough for practical deployment due to some configuration and stability concerns. The first concern is the configuration complexity of PI control-based AQMs. Configuration of PI control-based AQMs involves determining at least two parameters, a proportional parameter (K_P) and an integral parameter (K_I), in order to work effectively over a wide range of traffic conditions. Silva et al. [5] present a complicated process for configuring a PI controller for a system with time-delay but lack of expert knowledge of control theory may render the controller ineffective. The second concern is the discretization in PI control-based AQMs. The discretization of a continuous time-domain control function introduces a state measurement interval, such as the queue-sampling epoch in the PI AQM [3], that affects the stability of the system. However, this measurement interval is often neglected in the stability analysis adding uncertainty to the already complicated controller configuration process. The last concern is that recent TCP model-based AQM design approaches [2], [3], [4] have a small but fundamental stability issue in that a linear TCP model is used to analyze the range of stability for the control parameters without validating the linearity of the TCP system. That is, the linear models obtained from the non-linear stochastic TCP behavioral model may not accurately represent the behavior of the actual TCP congestion control system.

Based on sound understanding of PI control for Internet traffic and stability analysis using a linear TCP model, this section explores the possibilities of reducing the PI configuration parameters for active queue management, and presents the ARC logic for a configuration-optimized PI control algorithm for TCP congestion control systems. First, we design a rate-based algorithm that implements the general PI traffic controller algorithm in [3], enhancing the traffic rate estimation mechanism to provide an efficient and flexible controller configuration. Then, we carefully reduce the control parameters to obtain the ARC logic. Next, we model the ARC dynamics in a differential equation and use it along with the linear TCP behavior model from [7] to perform TCP-ARC system stability analysis, while also showing the effect of the state measurement interval on the system stability. Lastly, we validate the linearity of the TCP system as

Algorithm 1 Rate-Based PI Controller for AQM

Every d seconds (epoch):
1: $p \leftarrow p + \alpha(b - d\gamma C) + \beta(q - q_0)$;
2: $b \leftarrow 0$;
Every $packet$ arrival:
3: $b \leftarrow b + sizeof(packet)$;
4: $notify(packet, p)$;
Variables:
p: congestion notification probability
q: queue length in bytes
b: total bytes received this epoch
Parameters:
C: link capacity (bytes per second)
γ: target link utilization ($0 < \gamma \leq 1$)
q_0: target queue length in bytes
d: measurement interval
α: virtual queue control constant
β: queue control constant

well as the correctness and usefulness of the ARC-TCP system model through simulation.

Algorithm 1 shows our initial rate-based enhancement of the queue-based PI controller algorithm from [3], with added support for QoS configuration by introducing an additional target link utilization parameter (γ) in the PI control logic at *line 1*. When $\gamma = 1$, the rate-based PI control logic is identical to the queue-based control logic except for the incoming traffic amount measurement. However, when $0 < \gamma < 1$, the PI control logic can be regarded as maintaining a virtual link as in AVQ [4] and making control decisions proportional to the virtual queue length, where the virtual link capacity is dynamically adjusted to handle the traffic still present in the physical queue due to control error from the previous measurement interval (epoch). Thus, this version of PI first reserves a portion of the physical link capacity proportional to the queue displacement from the targeted length ($q - q_0$) and then determines the virtual capacity used to control aggregated traffic for the next epoch based on γ and the remaining physical capacity. This view of the PI control behavior is clearly shown when rewriting the logic in the following form:

$$p \leftarrow p + \alpha(b - \gamma(dC - \tfrac{\beta}{\gamma\alpha}(q - q_0)))$$

When the capacity reserved for the overfilled queue is proportional to the queue displacement, the necessary range condition for the proportional queue control parameter is:

$$0 < \tfrac{\beta}{\gamma\alpha} \leq 1$$

since the controller needs to reserve no more capacity than needs to be served for the upcoming epoch. We now consider the most conservative case of setting

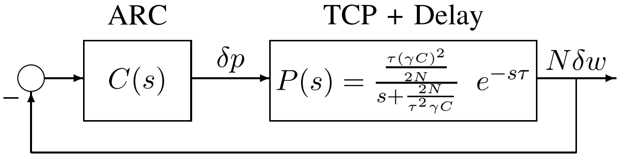

Fig. 1. TCP-ARC Feedback Control System with Transmission Delay

the proportional queue control parameter to one (i.e. $\beta = \gamma\alpha$). Then, the PI control logic is reduced to the following ARC logic:

$$p \leftarrow p + \alpha(b - \gamma(dC - (q - q_0))) \quad (1)$$

Next, we model the dynamics of the ARC logic for stability analysis. Figure 1 shows the block diagram of a TCP-ARC feedback control system that models N TCP sources and a single congested ARC router using the linear TCP model from [7], where τ is round trip time the system, C is the capacity of the congested link, and w is the expected TCP window size in packets given a congestion notification probability of p from the system. The system model also uses delta (δ) notation for each system variable to express the displacement from the equilibrium state. Thus:

$$\begin{aligned} \delta p &= p - p_0 \\ \delta w &= w - w_0 \\ \delta q &= q - q_0 \end{aligned}$$

where, p_0, w_0 and q_0 are the values of corresponding system variables at system equilibrium (assuming equilibrium exists). Using the number of packets received ($Nw(t)$) instead of the number of bytes received (b) to be compatible with the TCP model we use and the unit of link capacity (C in packets per second), the difference equation that models the dynamics of the ARC logic is:

$$\triangle p = \alpha(Nw(t) + \gamma(q(t) - q_0) - d\gamma C) \quad (2)$$

The ARC difference equation is transformed to use delta notation in order to adapt to the input and output of the TCP model given in delta format, and then converted into a differential equation:

$$\begin{aligned} \triangle\delta p &= \delta p - \delta p_{prev} = (p - p_0) - (p_{prev} - p_0) = \triangle p \\ &= \alpha(Nw - Nw_0 + \gamma(q - q_0) + Nw_0 - d\gamma C) \\ &= \alpha(N\delta w + \gamma\delta q + Nw_0 - d\gamma C) \\ \dot{\delta p} &= \lim_{\triangle t \to 0} \frac{\triangle\delta p}{\triangle t} \\ &= \lim_{\triangle t \to 0} \frac{\alpha(N\delta w + \gamma\delta q + Nw_0 - d\gamma C)}{\triangle t} \\ &\approx \frac{\alpha}{d}(N\delta w + \gamma\delta q + Nw_0 - d\gamma C) \quad (3) \end{aligned}$$

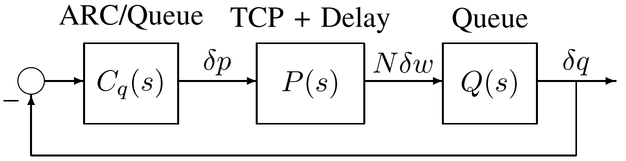

Fig. 2. TCP-ARC System with the Queue Model Removed from the ARC Transfer Function

Note in the last step of the above discrete to continuous time domain conversion that the traffic state measurement interval (d) uses an approximation for $\lim_{\triangle t \to 0} \triangle t$. This approximation, also used in the discretization processes, is valid for sufficiently small d. Note also that the ARC differential equation has a control parameter represented by $\frac{\alpha}{d}$ that needs to be configured for system stability and responsiveness. Once the stable range of $\frac{\alpha}{d}$ is found, α can be determined by choosing a sufficiently small d value. By applying a Laplas transformation to the ARC differential equation (Equation 3), the transfer function of ARC ($C(s)$) becomes:

$$C(s) = \frac{\frac{\alpha}{d}\left(s + \frac{1+\gamma}{\tau}\right)}{s\left(s + \frac{1}{\tau}\right)} \quad (4)$$

Before continuing our stability analysis, in order to identify the controller type, we take the queue behavior model embedded in the ARC model out from the ARC transfer function such that $C(s) = Q(s)C_q(s)$. Figure 2 represents this view of the system. Now:

$$\begin{aligned} Q(s) &= \frac{\frac{1}{\tau}}{s + \frac{1}{\tau}} && (5) \\ C_q(s) &= \frac{\alpha(1+\gamma)}{d} \frac{\frac{\tau s}{1+\gamma} + 1}{s} && (6) \\ &= K_I \frac{\frac{s}{T_p} + 1}{s} \end{aligned}$$

Equation 6 indicates that the queue-based implementation of ARC is a special PI controller that fixes the constant $T_p = \frac{1+\gamma}{\tau}$ such that the proportional constant ($K_P = \frac{K_I}{T_p}$) and the integral constant (K_I) of the compensator has the following relationship:

$$K_P = K_I \frac{\tau}{1+\gamma} \quad and \quad K_I = \frac{\alpha(1+\gamma)}{d}$$

We next perform frequency response analysis on the TCP-ARC system model, applying Bode stability criteria [8] to find the stable operating range of the $\frac{\alpha}{d}$ parameter for a chosen range of traffic conditions. The Bode method evaluates the stability of a closed-loop system by examining the open-loop system response to sinusoidal inputs with various frequencies. The open-loop transfer function of TCP-ARC system is:

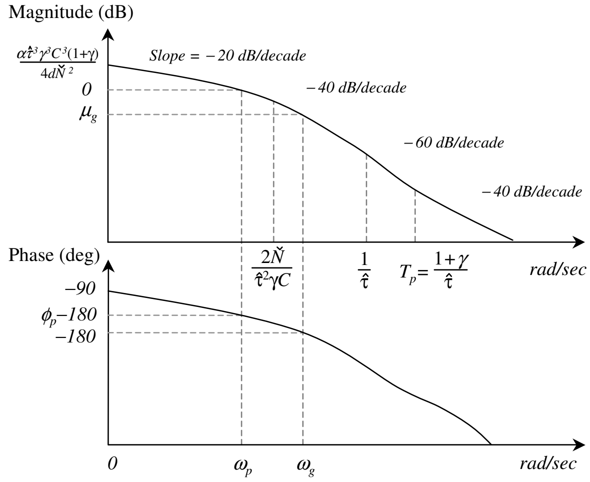

Fig. 3. Bode Plot of TCP-ARC System

$$G(s) = C(s)P(s) = \frac{\frac{\alpha\tau^3\gamma^3C^3(1+\gamma)}{4dN^2}\left(\frac{\tau}{1+\gamma}s + 1\right)e^{\tau s}}{s\left(\frac{\tau^2\gamma C}{2N}s + 1\right)(\tau s + 1)} \quad (7)$$

The two open-loop system responses the Bode method uses are the magnitude gain in decibels ($20\log_{10}|G(j\omega)|$) and the phase shift ($\angle G(j\omega)$) of the output sinusoid given as functions of input sinusoid frequency, which for the TCP-ARC system are computed as follows:

$$\begin{aligned} \mu(\omega, \tfrac{\alpha}{d}) &= 20\log_{10}|G(j\omega)| \\ &= 20\log_{10}\left(\frac{\frac{\alpha\tau^3\gamma^3C^3(1+\gamma)}{4dN^2}\sqrt{\left(\frac{\tau\omega}{1+\gamma}\right)^2+1}}{\omega\sqrt{\left(\frac{\tau^2\gamma C\omega}{2N}\right)^2+1}\sqrt{(\tau\omega)^2+1}}\right) \\ \phi(\omega) &= \angle G(j\omega) \\ &= \tan^{-1}\left(\frac{\tau\omega}{1+\gamma}\right) - \tan^{-1}\left(\frac{\tau^2\gamma C\omega}{2N}\right) \\ &\quad - \tan^{-1}(\tau\omega) - \frac{180^\circ}{\pi}\tau\omega - 90^\circ \end{aligned}$$

Figure 3 shows a Bode plot for an example TCP-ARC system. Before discussing the characteristics of the TCP-ARC system described by the Bode plots, we briefly introduce the Bode stability criteria that states the following: a closed-loop system is stable when the magnitude gain of the open-loop system is less than $-6dB$ for the input frequency that causes a 180° phase lag to the output signal, and also the phase lag is in between 30° and 60° for the input frequency that results in zero magnitude gain. That is, $\mu_g < -6$ and $30^\circ < \phi_p < 60^\circ$ in Figure 3 where:

$$\left\{ \begin{array}{lcl} \mu_g &=& \mu(\omega_g, \frac{\alpha}{d}) \\ \phi_p &=& \phi(\omega_p) + 180^\circ \end{array} \right. \quad \left\{ \begin{array}{lcl} \omega_g &=& \phi^{-1}(-180^\circ) \\ \omega_p &=& \mu^{-1}(0, \frac{\alpha}{d}) \end{array} \right.$$

Using Bode stability criteria to configure a general PI controller, such as in [3], with two control parameter involves deformation of both the magnitude and phase curves as the parameter values change, adding complications. And unfortunately, most other available PI tuning methods are not valid for systems with time delay. Moreover, the few available PI tuning methods for time-delayed systems may require expert decisions and only work for first-order systems [5]. ARC eases the configuration problems for TCP systems by fixing the controller constant $T_p = \frac{1+\gamma}{\hat{\tau}} > \frac{1}{\hat{\tau}}$. This results in the phase shift response curve of the TCP-ARC system that is not a function of the $\frac{\alpha}{d}$ parameter, and thus fixes the phase shift curve during the tuning process. In addition, the magnitude gain curve only moves vertically without deformation as a function of $\frac{\alpha}{d}$, easing the tuning process.

Given the link capacity (C) and the target utilization (γ), the stability boundary condition of the TCP-ARC system is characterized by the minimum expected number of TCP flows ($\check{N}$) and the maximum expected round-trip time of the system ($\hat{\tau}$). These parameters determine the location of the zero and poles of $G(s)$, characterizing the system into one of the following three cases:

$$\begin{aligned} Case\ 1 &: \frac{2\check{N}}{\hat{\tau}^2\gamma C} < \frac{1}{\hat{\tau}} < \frac{1+\gamma}{\hat{\tau}} \\ Case\ 2 &: \frac{1}{\hat{\tau}} \le \frac{2\check{N}}{\hat{\tau}^2\gamma C} < \frac{1+\gamma}{\hat{\tau}} \\ Case\ 3 &: \frac{1}{\hat{\tau}} < \frac{1+\gamma}{\hat{\tau}} \le \frac{2\check{N}}{\hat{\tau}^2\gamma C} \end{aligned}$$

$Case\ 1$: This system depicted by Figure 3 has phase lags of $\phi(0^+) = -90^\circ$ and $\phi(\frac{1}{\hat{\tau}}) < -180^\circ$ indicating that at least one $\omega_g < \frac{1}{\hat{\tau}}$ exists.

$$\begin{aligned} \phi\left(\tfrac{1}{\hat{\tau}}\right) &= \tan^{-1}\left(\frac{\hat{\tau}}{1+\gamma}\,\frac{1}{\hat{\tau}}\right) - \tan^{-1}\left(\frac{\hat{\tau}^2\gamma C}{2\check{N}}\,\frac{1}{\hat{\tau}}\right) \\ &\quad - \tan^{-1}(1) - \frac{180^\circ}{\pi} - 90^\circ \\ &= (45^\circ - \epsilon_1) - (45^\circ + \epsilon_2) - 45^\circ - \frac{180^\circ}{\pi} - 90^\circ \\ &< -180^\circ \end{aligned}$$

Further examining the shape of $\phi(\omega)$ using the derivative $\frac{d\phi(\omega)}{d\omega}$ reveals that $\phi(\omega)$ is a decreasing function of ω except for a possible small mound in the neighborhood of $\frac{1+\gamma}{\hat{\tau}}$. However, since $\frac{1}{\hat{\tau}} < \frac{1+\gamma}{\hat{\tau}}$, the smallest (and usually the only) $\omega_g < \frac{1}{\hat{\tau}}$ with $\frac{d\phi(\omega)}{d\omega}|_{\omega<\omega_g} < 0$ can always be found. This, plus the fact that the magnitude gain curve $\mu(\omega, \frac{\alpha}{d})$ is a strictly decreasing function of ω starting from a positive value at $\omega = 0^+$, guarantees that it is always possible to find a range of $\frac{\alpha}{d}$ that makes $\omega_p(\frac{\alpha}{d}) < \omega_g$ and thus have a positive ϕ_p. If a TCP-ARC system can ever be stabilized for a given $\check{N}$ and $\hat{\tau}$ boundary, one should be able to refine the range of $\frac{\alpha}{d}$ such that $30^\circ < \phi_p < 60^\circ$. A useful necessary condition for the system stability is $\mu(\frac{1}{\hat{\tau}}, 0) < -6$.

$Case\ 2$ and $Case\ 3$: The same analysis used for a $Case\ 1$ system can be applied to $Case\ 2$ and $Case\ 3$ systems with two differences. The first difference for both $Case\ 2$ and $Case\ 3$ systems is that $\phi(\frac{2N}{\hat{\tau}^2\gamma C}) < -180^\circ$. The second difference for $Case\ 3$ systems is that $\phi(\omega, \frac{\alpha}{d})$ is a strictly decreasing function of ω. It is always possible to find a range $\frac{\alpha}{d}$ that makes $\omega_p(\frac{\alpha}{d}) < \omega_g$ and have a positive ϕ_p, where a necessary condition for system stability is $\mu(\frac{2N}{\hat{\tau}^2\gamma C}, 0) < -6$.

For all cases, we can determine if a TCP-ARC system can be stabilized for the chosen $\check{N}$ and $\hat{\tau}$ boundary, and find the stable operating range of $\frac{\alpha}{d}$ in two steps:

$$\begin{aligned} &\{\tfrac{\alpha}{d} \mid \mu(\omega_g, \tfrac{\alpha}{d}) < -6\} \quad and \\ &\{\tfrac{\alpha}{d} \mid -150^\circ < \phi(\omega_p(\tfrac{\alpha}{d})) < -120^\circ\} \end{aligned} \tag{8}$$

Thus, ARC configuration involves choosing a reasonable minimum number of flows ($\check{N}$) and maximum expected round-trip time ($\hat{\tau}$) for the system we wish to support, finding the $\frac{\alpha}{d}$ range that satisfies the first condition of Equation 8, and refining the range to satisfy the second condition of Equation 8. Once the stable range is found, choosing a sufficiently small d gives the range of α. The ARC configuration issues of choosing reasonable $\check{N}$ and $\hat{\tau}$ stability boundary conditions and the measurement interval d is discussed in Section III.

We implemented the ARC algorithm shown in Algorithm 2 in NS[1] and designed experiments to validate linearity of the TCP system and verify the correctness of our model. Linearity of a system can be validated via simple frequency analysis by introducing a sinusoid into the system. If a system has linearity, the output of the system should also be a sinusoid with a possibly altered magnitude and frequency. We artificially injected a sinusoid into a simulated TCP-ARC system representing the incoming traffic rate estimation error at the congested ARC router and measured the system throughput. Then, we compared the simulated throughput with the output of the analytic TCP-ARC system model.

We used a dumbbell topology for the simulations, uniformly varying the round-trip times of 35 FTP-TCP connections from $90ms$ to $110ms$. We set the physical queue length to 500 packets, where the system used 1 Kbyte packets. For the congested-link, we used $C = 8$

[1]The NS homepage is at http://www.isi.edu/nsnam/ns/

Algorithm 2 Aggregate Rate Controller

Every d seconds:

```
p ← p + α(b − γ(dC − (q − q_0)));
b ← 0;
```

Every *packet* arrival:

```
if (uniform(0, 1) ≤ p) then
    if (mark(packet) == false) then
        drop(packet);
        return;
    end if
end if
if (enqueue(packet) == false) then
    drop(packet);
    return;
end if
b ← b + sizeof(packet);
```

Functions:

$mark(packet)$: ECN mark the packet. Return $false$ on error.
$enqueue(packet)$: Enqueue the packet. Return $false$ if no room.
$drop(packet)$: Drop the packet.

Variables:

p, q, b

Parameters:

C: link capacity (bytes per second)
γ: target link utilization ($\gamma = C_0/C$)
q_0: target queue length in bytes
d: measurement interval
α: TCP congestion feedback constant

Mbps once with the target utilization $\gamma = 0.95$, and once with $\gamma = 0.50$. The ARC control parameters were set to $d = 200$ ms and $\alpha = 0.112$ for the simulations. For the ARC rate estimation error, we use a low frequency sinusoid of $0.5\sin(0.1(t-50))$ Mbps, and a high frequency sinusoid of $0.5\sin(100(t-50))$ Mbps.

Figure 4 compares the throughput of the simulated systems with that of our analytic model for the two sinusoidal inputs (top and bottom) under the two different target loads (γ). The simulated results closely match the output of the TCP-ARC analytical model, verifying the linearity of the TCP congestion control system and also the correctness of our TCP-ARC model. In addition, the ability to meet custom traffic loads illustrates the configuration flexibility of ARC. More importantly, the resilient TCP-ARC congestion control system performance for the low-frequency and high-frequency traffic rate estimation errors indicates potentially resilient ARC performance in the presence of TCP-unfriendly, unresponsive and/or short Web traffic.

III. ARC Configuration

In the previous section, we have shown that the ARC control parameter $\frac{\alpha}{d}$ can be found to stabilize a TCP system characterized by $\check{N}$ and $\hat{\tau}$ boundary conditions. This

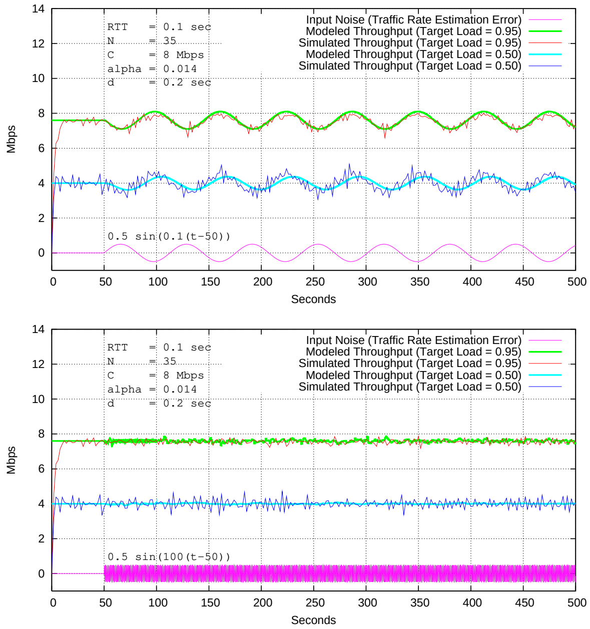

Fig. 4. TCP-ARC Model Validation: Throughput of two systems - low-frequency (top) and high-frequency (bottom) traffic rate estimation errors are artificially injected into ARC.

section addresses the configuration issue of selecting the boundary condition epoch length (d) to enable ARC to be resilient over a wide range of traffic conditions.

Although a PI controller considerably broadens the range over which a controller with a given configuration can be stable [3], fundamental configuration issues still remain for ARC or any other PI controll-based AQM. Consider the case where ARC is configured for an average-case traffic scenario with a relatively large number of flows ($\check{N}$) and a typical round-trip time $\hat{\tau}$. If there are fewer TCP flows than average and/or the average round-trip time is unusually large, the resulting TCP-ARC system may become instabile. We will refer to this configuration as an "overshoot problem". On the other hand, if ARC is configured to support a worst-case scenario with a small number of flows ($\check{N}$) and a large round-trip time ($\hat{\tau}$), the resulting system will have a significantly larger response time to changes in congestion when there is the average-case traffic. Thus, the system will suffer from significant queuing delays and queue overflows before adapting to any changing traffic conditions. We will refer to this configuration as an "undershoot problem".

Most TCP flows have their maximum bitrates limited by one or more of: the capacity of the ISP access points or local networks; the maximum TCP window sizes; and the sizes of the objects being downloaded [9]. Therefore, the overshoot problem is not a significant concern for many core routers since backbone routers are not likely to be congested by a small number of TCP flows. Also flows that traverse the backbone typically have a relatively low average round-trip time [9], [10], making them unable to create an overshoot problem. However, the overshoot problem may occur in enterprise or access network routers as the traffic load and round-trip time may vary significantly over the course of a day. In fact, the overshoot problem is more critical than the undershoot problem since it can severely degrade network stability even under light traffic loads. At the same time, it is desireable to avoid undershoot problems since that deteriorates the potential benefits of AQM.

We address the overshoot and undershoot configuration problems for ARC by carefully interpreting the meaning of applying the system stability analysis from Section II to practical network operations. Finding a reasonable $\check{N}$ and $\hat{\tau}$ for the $\frac{\alpha}{d}$ configuration starts from the realization that it is inadequate to apply stability determined through our stochastic system model to a systems with a small number of TCP flows and a large average round-trip time (a fragile system). That is, no matter how well we tune ARC using the model, it is impossible for an AQM to achieve a high link utilization and low queuing delay for a fragile system since the aggregate traffic rate fluctuates due to the bursty characteristics of TCP. Therefore, tuning ARC, or any other PI control-based AQM, for a fragile system is an ineffective use of the controller capability and it may cause unnecessary queuing delays and overflows for normal traffic loads. Thus, the ARC configuration objective is to effectively manage average traffic conditions while avoiding a large overshoot problem that could significantly degrade network stability.

In order to determine when an overshoot threatens stability for the fragile lower bound of a system we wish to support, denoted by $\check{N}_{min}$ and $\hat{\tau}_{max}$, we need to determine the sustainable congestion notification probability estimation error ($\delta p_s < 1$) in order for the lower bound system to be least operational. We develop simple frequency analysis on a TCP plant ($P(s)$) to quantify δp_s. The least operational state of a TCP system is the following: a TCP system is least operational if for congestion notification errors in the range [$-\delta p_s$, δp_s] introduced to the TCP plant, the output range of the system is [$-\gamma C$, γC]. The equivalent mathematical representation is:

$$|P(j\omega)| \;=\; \frac{\frac{(\hat{\tau}_{max})^3\gamma^3 C^3}{4(\check{N}_{min})^2}}{\sqrt{\omega^2\left(\frac{(\hat{\tau}_{max})^2\gamma C}{2\check{N}_{min}}\right)^2+1}} \;\leq\; \frac{\gamma C}{\delta p_s}$$

This inequality always holds for high frequency inputs. For low frequency errors in the congestion notification probability $\left(\omega \ll \frac{2\check{N}_{min}}{(\hat{\tau}_{max})^2\gamma C}\right)$, we have:

$$\delta p_s \;\leq\; \frac{4(\check{N}_{min})^2}{(\hat{\tau}_{max})^3\gamma^2 C^2}$$

Applying this statement to the TCP-ARC system, if ARC makes congestion estimation errors within [$-\delta p_s$, δp_s] for the fragile lower bound of the system we wish to support, the system will be least operational. In order to accomplish this, it is necessary to set the $\frac{\alpha}{d}$ parameter such that ARC increases p sufficiently less than δp_s during one $\hat{\tau}_{max}$ interval:

$$\frac{\alpha}{d} \;\leq\; \frac{\alpha_{max}}{\hat{\tau}_{max}} \;\ll\; \delta p_s$$

This minimum stability condition can be used for ARC configuration to avoid threatening stability with a large overshoot problem for the fragile boundary system we choose to support. We first choose an average TCP-ARC system to stably support, described by $\check{N}$ and $\hat{\tau}$, and use the ARC configuration guidelines introduced in Section II to find the stable $\frac{\alpha}{d}$ range. Then, we choose the fragile boundary system described by $\check{N}_{min}$ and $\hat{\tau}_{max}$, and use the minimum stability condition to verify that the parameter range is safe for the fragile boundary system.

Next, we need to determine the measurement epoch d to complete the configuration. The stability of backbone routers may not be sensitive to the choice of d, but proper selection of d is critical for enterprise or access network router configurations. The stochastic TCP system model may not accurately depict the stability of the system with a large average round-trip time and a small number of TCP flows since packets would tend to arrive at the router in bursts, breaking the assumption of stochastic packet arrivals. However, we can relax the stochastic assumption for the TCP-ARC system by choosing a sufficiently large d, larger or at least equal to the maximum average round-trip time we wish to support ($\hat{\tau}_{max}$). In this way, the effect of the traffic bursts on the ARC control decision can be minimized, reducing the noise in the incoming traffic rate estimation caused by the traffic bursts. However, choosing too large a d will affect the system responsiveness, weakening the small interval assumption made for the algorithmic discretization. As a compromise, we recommend setting the measurement epoch to the maximum expected round-trip time, or $d = \hat{\tau}_{max}$. According to a recent Internet measurement study [9], the median of typical median round-trip times is less than 1 second, and 90% of the median round-trip times are under 2 seconds. Therefore, depending on the targeted level of network resilience and the specific traffic characteristics of the network, setting $\hat{\tau}_{max}$ in between 1 and 2 seconds is appropriate for most enterprise or access network routers. For backbone routers, selecting an even smaller $\hat{\tau}_{max}$ to enhance the granularity of congestion control may also be effective.

Summarizing our analysis for configuring ARC over a wide range of traffic loads, we present the final guidelines that are recommended along with Equation 8:

$$d = \hat{\tau}_{max} \quad and \quad \alpha \ll \frac{4(\check{N}_{min})^2}{(\hat{\tau}_{max})^3\gamma^2 C^2} \tag{9}$$

IV. Evaluation

This section compares the performance of ARC with that of PI [3], AVQ [4], SFC [2] and Drop-Tail through detailed simulations varying number of flows, round-trip times and bottleneck link capacity, with competing background Web traffic and 2-way traffic (which can result in ack compression). We also simulate Web flash crowds to stress-test the AQM controllers and multiple congestion bottlenecks to provide a more realistic Internet traffic environment. For all evaluations, we use the IP packet simulator NS. NS includes source code for PI and AVQ and we implement SFC based on [2] and extend NS to support ARC.[2]

Unless otherwise noted, a dumbell network topology is used with a bottleneck link capacity of 10 Mbps and a maximum packet size of 1000 bytes. Round-trip link delays are randomly uniformly distributed over the range [60:1000], based on measurements in [9]. The physical queue limit for each AQM and the drop-tail queue is set to 500 Kbytes, which is approximately equal to the bandwidth-delay product for the mean round-trip time.

The settings for the parameters of the various AQMs are based on the recommendations by their authors. The target utilization, γ, for AVQ is set to 0.98 and the damping factor, α, is set to 0.15 according to Theorem 1 in [4]. The parameters for SFC are $k_1 = 0.0005$ and $k_2 = 0.2$, as used in [2]. The parameters for PI are $\alpha = 1.822 \times 10^{-5}$, $\beta = 1.816 \times 10^{-5}$ and sampling frequency $w = 170$, as in [3]. The settings for ARC have the measurement epoch $d = 1$ seconds, $\alpha = 1.42 \times 10^{-5}$, the target utilization $\gamma = 0.98$ and the target queue $q_0 = 0$. The ARC control parameters are chosen in order to effectively support the system with $\hat{\tau} = 500ms$ and $\check{N} = 50$, and to satisfy the minimum stability condition assuming the fragile boundary system has $\hat{\tau}_{max} = 2sec$ and $\check{N}_{min} = 10$.

In all simulations, we use ECN enabled NewReno TCP for both long-lived FTP flows and short-lived Web sessions. Each simulation has a number, specified in each experiment subsection, of forward and backward direction bulk transfer FTP flows. Also, each simulation has 300 background Web-like sessions (using the Webtraf code built into NS) that start evenly distributed during the first 30 seconds. Based on results from [11], [12], each Web session requests pages with 2 objects drawn from a Pareto distribution with a shape parameter of 1.2 and an average size of 5 Kbytes. The Web sessions have an exponentially distributed think time with a mean of 7 seconds, which results in an average utilization of 0.25 of the 10 Mbps capacity, a typical fraction of Internet traffic as reported by [13].

[2]The ARC code is available at http://perform.wpi.edu/downloads/-arc/.

A. *Long-Lived TCP Flows*

This experiment compares the performance of ARC, PI, AVQ, SFC and Drop-Tail over a range of traffic loads with long-lived TCP flows. Each simulation begins with 10 forward direction FTP flows with start times uniformly randomly distributed over [0:50] seconds, and is accompanied by 300 background Web sessions and 50 backward direction FTP flows. After 200 seconds, an additional 40 FTP flows are added with start times uniformly randomly distributed over the subsequent 50 seconds. The total number of FTP flows doubles every 200 seconds thereafter, resulting in 100 flows at time 400, 200 flows at time 600, 400 flows at time 800 and 800 flows at time 1000. For each new interval, the FTP start times are uniformly randomly distributed over a 50 second range. We ran ARC twice, once with the settings specified earlier ($d = 1$ and $\alpha = 1.42 \times 10^{-5}$) and then with $d = 2$ (and $\alpha = 2.84 \times 10^{-5}$ to preserve the α/d ratio), in order to test the sensitivity of ARC to the rate measurement interval.

Figure 5 (top) depicts the queue dynamics for the four AQMs with drop-tail (DT) shown as a reference. Drop-tail exhibits the expected large queue oscillations when there are few flows and stable, but large, queue sizes when there are many flows. SFC has stable queue dynamics for all numbers of flows. However, without an integral control component, SFC exhibits a steady increase in the average queue size as the number of flows increases, approaching the physical queue limit when there are 800 FTP flows. AVQ exhibits unstable queue dynamics when there are few flows but stabilizes with relatively low queues once there are 200 or more FTP flows. PI and ARC ($d = 1$ and $d = 2$) are similar, with stable queue dynamics for all numbers of flows, and with only short-term queue size increases each time a large group of flows arrives. Comparing ARC with $d = 1$ to ARC with $d = 2$ shows that ARC is not very sensitive to the selection of the measurement interval.

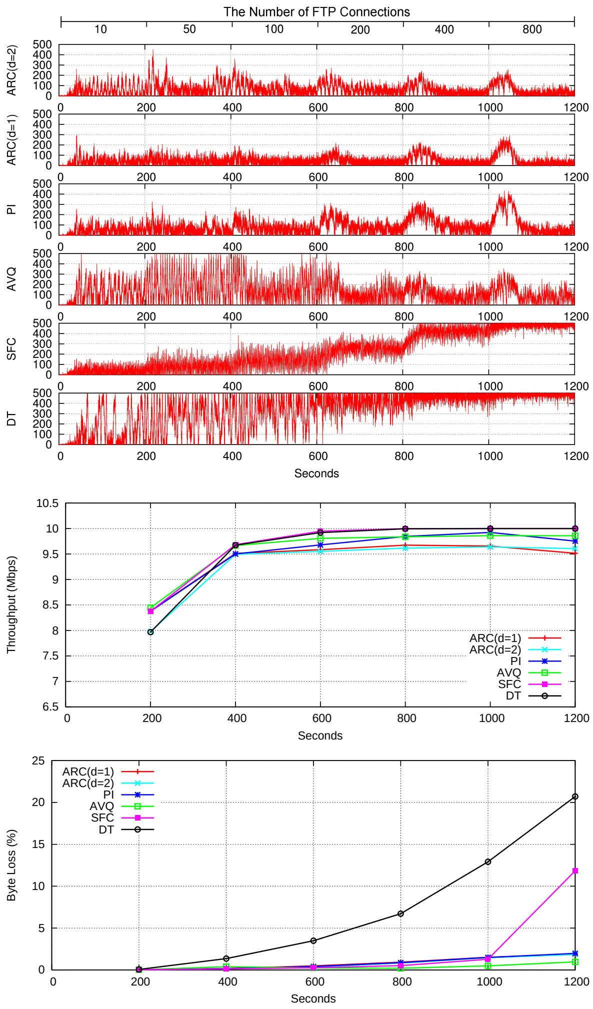

Fig. 5. Network Statistics: Increasing the number of FTP flows

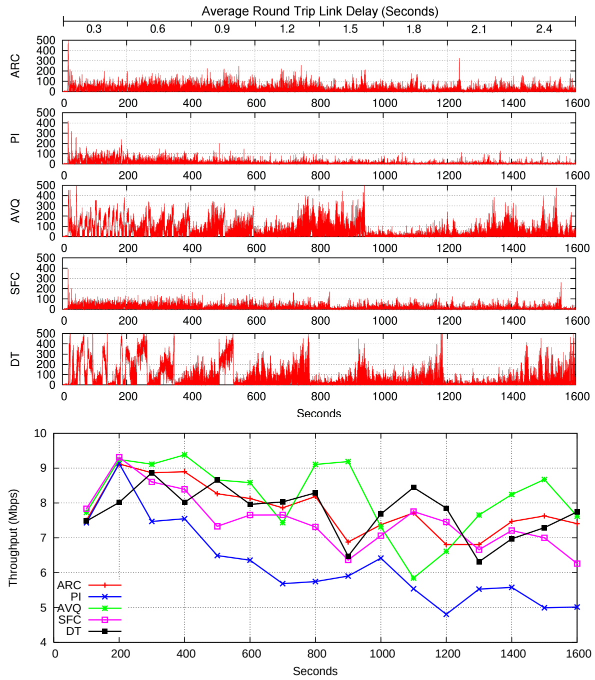

Fig. 6. Network Statistics: Increasing the average round-trip time

Figure 5 (middle) depicts the throughput for the AQMs and drop-tail. Once the number of FTP flows is 50 or more, each queue management scheme is able to obtain a throughput of more than 9.5 Mbps. With more than 200 FTP flows, drop-tail has the highest throughput and SFC has a throughput nearly equal to that of drop-tail. However, drop-tail has the lowest utilization when there are only 10 FTP flows. For 50+ FTP flows, AVQ and ARC keep their throughput close to their target utilization, and PI and AVQ achieve nearly the same throughput for all numbers of flows. Overall, PI and AVQ have less control over the queue length than does ARC with a correspondingly slightly higher throughput than ARC.

Figure 5 (bottom) depicts the loss rates, where most of the losses seen for AQMs at the lower range of traffic load are due to packets that cannot be ECN marked such as the reverse traffic acknowledgments. The figure verifies that AQM can significantly reduce data losses for ECN traffic over drop-tail queue management as long as AQM can avoid or minimize buffer overflows. However, when there are consistent buffer overflows, as in SFC for high traffic loads, the benefits of AQM quickly diminish. A case where an AQM (AVQ, in this case) can incur even greater data losses than drop-tail is illustrated in Section IV-C.

B. *Round-Trip Time*

The next experiment illustrates behavior of the different queue mechanisms over a range of round-trip times, from about 0.3 to 2.4 seconds. In order to simulate the effects of a range of round-trip time in one experimental run, we gradually increase the round trip link delay (RTLD) of the congested link by 300 ms every 200 simulation seconds. We have run multiple sets of simulations, all with 300 background Web sessions but a different numbers of forward and backward FTP flows to observe the effect of round-trip time on different levels of load. For brevity, we show the results for 5 forward FTP flows and 10 backward FTP flows, but the overall performance trends are similar for other numbers of flows.

The queue dynamics in Figure 6 (top) show that ARC, PI and SFC keep queue size low and dampen queue oscillations, while AVQ and drop-tail have poor control over the queue. The throughput comparisons in Figure 6 (middle) show that PI under-utilizes link capacity far more than the other queue management schemes. This inefficiency is due to PI's method of determining the traffic rate by sampling the queue with high frequency. When the system becomes fragile and the traffic arrives in bursts, the frequent queue samples introduce a large estimation noise in the estimation of the incoming traffic rate. ARC and SFC[3] avoid this noise by directly measuring traffic rate over 1 second intervals and so can more accurately estimate incoming traffic rate than can PI. AVQ's packet-paced method of traffic rate measurement is also compatibly accurate.

C. Congested Link Capacity

For the next set of simulations, we increase the bottleneck link capacity initially set to 10 Mbps by 10 Mbps every 200 simulation seconds up to 50 Mbps, with 200 forward and 50 backward direction FTP flows and 300 background Web sessions. During the capacity increases, we do not change the AQM parameter settings. This has two effects: increasing the capacity makes the system more fragile and also brings out the undershoot problem (described in Section III) of an AQM slowly responding to network congestion. Thus, this experiment illustrates control parameter sensitivity of ARC, PI, AVQ and SFC.

Figure 7 illustrates the queue dynamics, throughput and data loss rates. AVQ is very sensitive to the initial control parameters. While AVQ is able to maintain throughput close to its target utilization, it does not effectively control queue oscillations and incurs even greater data losses than does drop-tail queue management.

PI again produces under-utilization and a sluggish response as the capacity increases, showing another drawback of queue-based rate estimation. As the traffic load decreases due to the capacity increase, PI cannot efficiently estimate the traffic rate, since the queue oscillations tell little about the underlying traffic rate.

Both ARC and SFC effectively control queue oscillation and achieve high throughput over the range of traffic loads. Yet, when the link capacity increases to 40 Mbps and over, SFC is not able to achieve as high a throughput as ARC due to its traffic load dependent control behavior. However, SFC is still able to achieve throughput at about the level of drop-tail.

D. Web Flash Crowd

In this experiment, we stress-test the AQMs with a realistic Web flash crowd. For this simulation, we have 25 forward and 50 backward direction FTP flows as background traffic, and an initial 300 Web sessions. After

[3]SFC [2] does not specify how its rate estimation mechanism should be implemented nor recommend a range of values for the measurement epoch. We used ARC's rate estimation mechanism to implement SFC in NS, and set the measurement epoch to 1 second as in ARC.

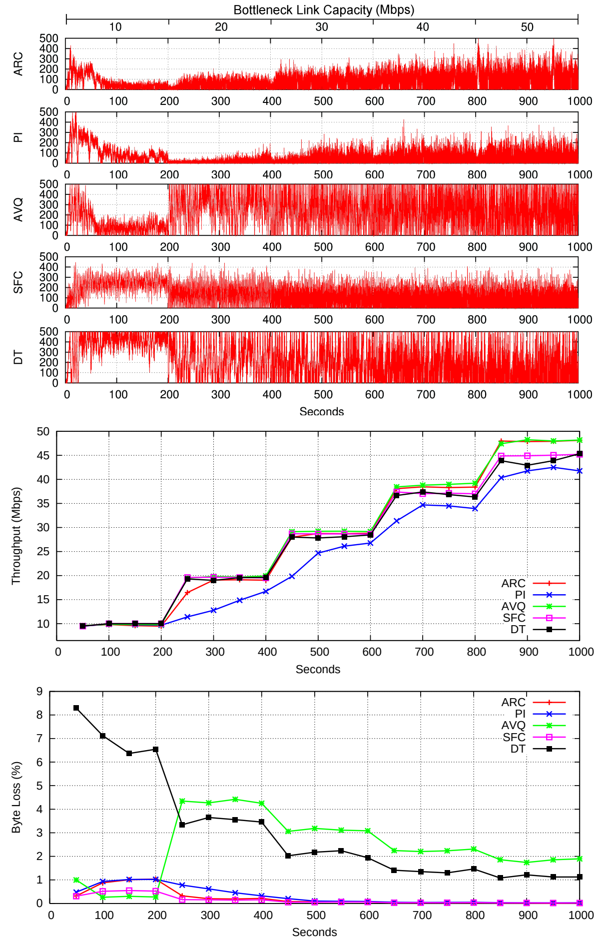

Fig. 7. Network Statistics: Increasing the bottleneck link capacity

a warm up period of 100 seconds, 1000 more Web sessions are uniformly randomly injected during the first 6000 simulation seconds and then detached during the next 6000 simulation seconds, giving a Web flash rate of 10 sessions per minute. Thus, we have a peak load of 1300 Web sessions, providing an offered load of about 1.35.

We use the Web flash rate of 10 sessions per minute based on the peak Web flash rate seen during FIFA World Cup 98 [11]. The peak flash rate (both increase and decrease) went from 2 to 10 million requests per hour in 2 hours during the France-Croatia game. This means an acceleration of about 1,110 $requests/min^2$ from the minimum object request rate of 33,333 $requests/min$ (= 2 million $requests/hour$).

The initial 300 Web sessions in our simulation offer an average utilization of about 0.25, a typical fraction of Internet traffic reported in [13], with the minimum object request rate of 5,143 $requests/min$ (300 $\times$ 2 $objs/click$ $\times$ 60/7 $clicks/min$). This is about 15% of the FIFA 98 base request rate. In order to get the proportional acceleration rate for our simulated minimum request rate,

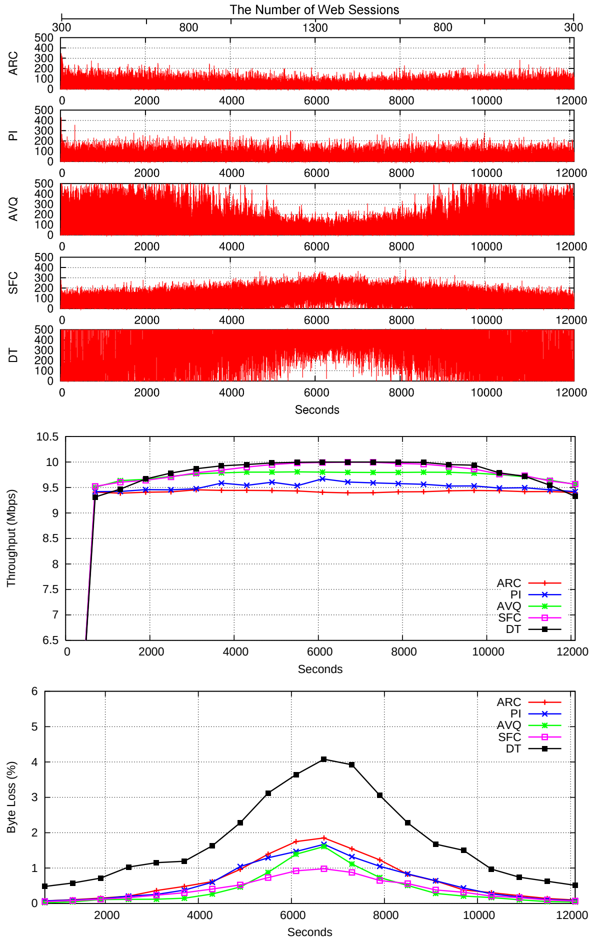

Fig. 8. Network Statistics: Web flash crowd

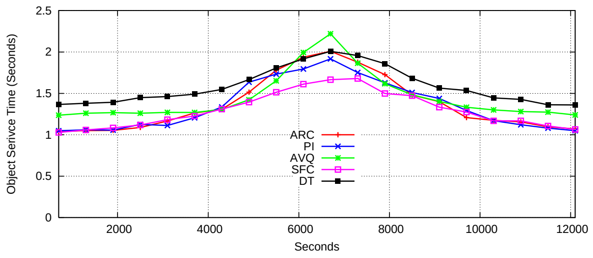

Fig. 9. Average Service Time: Objects less than 12 Kbytes

we take 15% of the FIFA 98 acceleration rate, that is about 170 $requests/min^2$ or 10 sessions per minute, and use it as our flash rate.[4]

Figure 8 depicts the network statistics including queue dynamics, throughput and data loss rate. In order to better understand the performance of the AQMs on Web traffic, we also analyzed the average object service time. Figure 9 plots average service time for objects less than 12 Kbytes (95% of the all generated Web objects), which

[4]We also tried the absolute flash rate of the FIFA 98 trace (1,110 $requests/min^2$ = 60 sessions per minute) and received simular overall results.

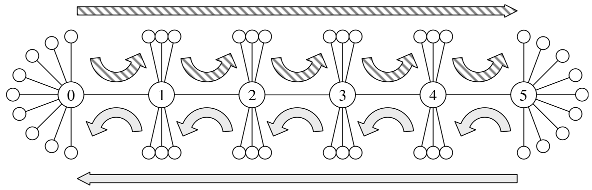

Fig. 10. Multiple Bottleneck Simulation: Setup

can be transmitted in about 4 round-trip times in the best case.

In general, the queue dynamics shown in Figure 8 (top) are very similar to those of Figure 5 (top). Likewise, Figure 8 (middle) shows that all AQMs achieve a throughput of around 9.4 Mbps or higher throughout the simulation. However, in Figure 8 (bottom), the data loss rates for AQMs are significantly increased (compared with Figure 5 (bottom)) for the traffic dominated by Web traffic due to the increased number of TCP-SYN packets that cannot by ECN marked. This illustrates that AQM gains for ECN traffic is significantly reduced for short-lived Web traffic.

Drop-tail performs the worst overall having the highest data losses, queuing delay and object service times. All AQMs performed well under the offered Web traffic load of 0.9 (before 4,000 seconds or after 10,000 seconds), except AVQ which has average service times, consistently higher than those of other AQMs because of its queue oscillations.

As the Web traffic load further increases over 0.9, SFC performs slightly better than other AQMs by stabilizing the queue higher than the other AQMs, thus achieving the overall highest throughput and the lowest data loss rate. ARC and PI perform very similarly in all performance aspects. Beyond the offered load of 0.9, AVQ starts to gain control over queue oscillations and performs comparably to ARC and PI.

E. Multiple Bottleneck Congestion

Figure 10 shows the network topology for the multiple bottleneck simulations. The simulated network has 5 bottleneck links (numbered 0 to 4) with capacities of 10 Mbps and a transmission delay of 20 ms. The edge links are 100 Mbps with 20 ms of transmission delay. Each striped arrow in Figure 10 represents 25 FTP flows plus 150 Web sessions, and each solid arrow represents 25 backward FTP flows.

Figure 11 depicts the queue dynamics of all congested links for ARC, PI, AVQ, SFC and drop-tail for the first 300 seconds. ARC has the best control over queue oscillations followed by PI, SFC and then AVQ. In

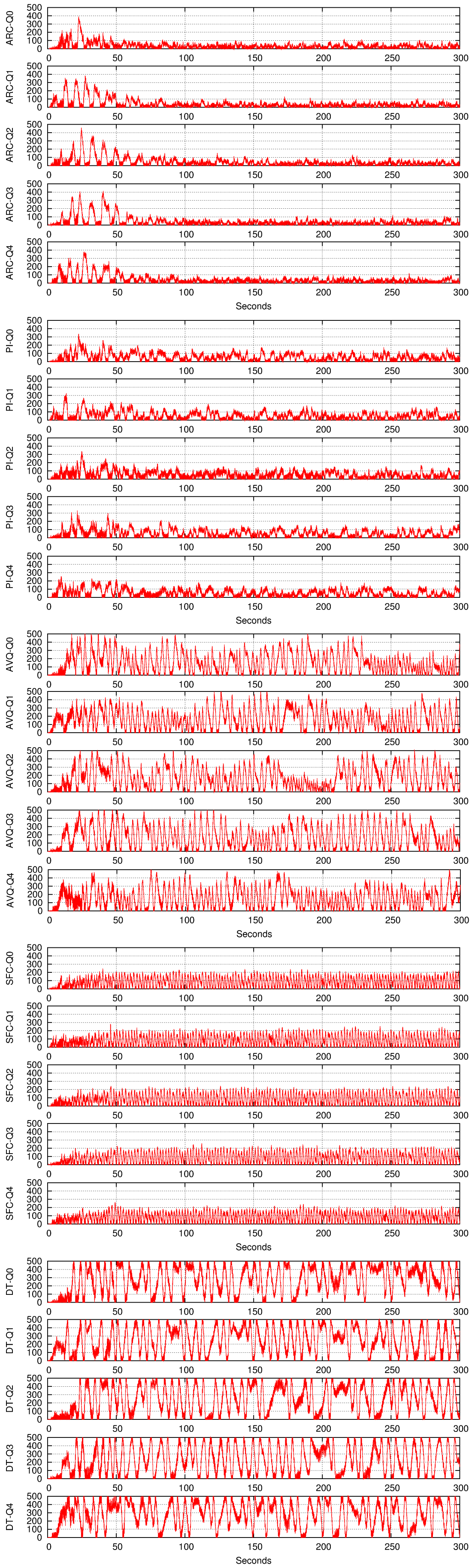

Fig. 11. Multiple Bottleneck Simulation: Queue dynamics

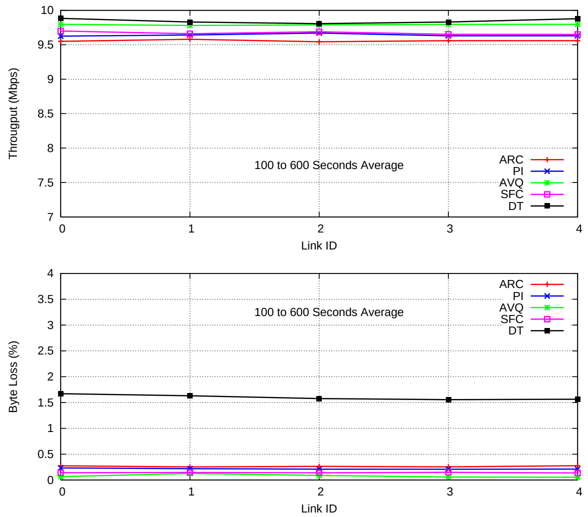

Fig. 12. Multiple Bottleneck Simulation: Throughput and loss

addition, all bottleneck queues are behaving stably and similarly to one another indicating that ARC and other AQM systems are globally stable. Figure 12 show the average throughputs and the average data loss rates of the bottleneck links, where all the AQM routers achieve a high throughput and a low data loss rate, consistent with previous single bottleneck analysis.

V. Extention to Support FAST

The ARC we have presented expects TCP with an additive increase multiplicative decrease (AIMD) source/network utility/congestion control protocol. This section discusses extending ARC to seamlessly support other similar utility control protocols, in particular, FAST [14] proposed to overcome the inefficiency of AIMD for high bandwidth-delay product networks.

Utility/congestion control protocols can either have a network centric control design, where network nodes explicitly determine utility (i.e., transmission rate) of individual traffic sources as in the eXplicit Congestion Protocol (XCP) [15], or have a distributed control design, where each individual traffic source determines its own utility using a demand function implicitly or explicitly chosen by the protocol and utility price of its network path implicitly or explicitly notified by the network. Most congestion control protocols proposed for the Internet, including various versions of TCP, TFRC [16], TCP Vegas [17] and FAST [14] are in the latter category.

Utility prices are determined based on a type of queuing delay along the network path: physical queuing delay for protocols with no explicit utility control support, and virtual queuing delay for protocols that assume the network provides the utility price. For example, in TCP

Algorithm 3 FAST-Extended ARC

Every d seconds:

$q_v \leftarrow q_v + (b - \gamma(dC - (q - q_0)))$;
$p_{tcp} = \alpha \times q_v$;
$p_{fast} = \phi^{-q_v/(\gamma C)}$;
$b \leftarrow 0$;

Every $packet$ arrival:

if $(typeof(packet) == fast)$ **then**
    $p = p_{fast}$;
else
    $p = p_{tcp}$;
end if
if $(uniform(0,1) \leq p)$ **then**
    if $(mark(packet) == false)$ **then**
        $drop(packet)$;
        $return$;
    end if
end if
if $(enqueue(packet) == false)$ **then**
    $drop(packet)$;
    $return$;
end if
$b \leftarrow b + sizeof(packet)$;

Functions:

$mark(packet)$: ECN mark the packet. Return $false$ on error.
$enqueue(packet)$: Enqueue the packet. Return $false$ if no room.
$drop(packet)$: Drop the packet.

Variables:

p, p_{tcp}, p_{fast}, q_v, q, b

Parameters:

C: link capacity (bytes per second)
γ: target link utilization ($\gamma = C_0/C$)
q_0: target queue length in bytes
d: measurement interval
α: TCP congestion feedback constant
ϕ: REM [6] communication constant shared by all FAST sources

or TFRC systems, utility price of a congested drop-tail link given in form of packet drop rate is determined implicitly by average queuing delay [18]. TCP Vegas, on the other hand, explicitly measures estimated queuing delay of congested drop-tail links in a network path for its utility price of the path. The utility price of a congested ARC or other PI-based AQM link, given in ECN marking probability, is computed based on the virtual queuing delay. Similarly, FAST, which has a demand control mechanism for traffic sources and a utility/congestion controller at network routers, uses virtual queuing delay over the virtual capacity at each link as the estimated price of the congested link.

The network utility price computation requirements being fundamentally the same for most distributed utility control protocols – a function of the network queuing delay – provides an opportunity for a single network to concurrently support similar distributed utility/congestion control protocols with little overhead. To illustrate this, we make a simple modification to the original ARC in Algorithm 2 to concurrently support FAST, which can be deployed in the current Internet without modification to IP specifications, as well as TCP.

Algorithm 3 shows the FAST-extended ARC, where q_v is an implementation of a virtual queue after physical queue error correction. The extended ARC uses $\alpha \times q_v$ as the link utility price for TCP traffic, which is basically the same implementation with the original ARC logic, and $q_v/(\gamma C)$ for FAST traffic. The only minor difference between this FAST utility price computation from [14] is that while the original FAST algorithm uses pure virtual queuing delay with no physical queue error control (i.e. uses an integral controller) this new version considers physical queue error control as well and uses a PI utility/congestion controller. Replacing the integral utility controller with the PI controller should have little effect on stability of the FAST system, since adding a proportional control, in general, does not significantly affect stability of a system.

For congestion notification, TCP uses a plain ECN marking or packet dropping communication scheme. In contrast, FAST uses REM [6] encoding/decoding, an alternative way to use the single ECN bit in the IP header for congestion communication. In Algorithm 3, ϕ is the REM communication constant shared by all FAST sources and FAST-enabled routers in the system.

Thus, by keeping track of the fundamentally compatible link utility prices with the support for the two different congestion communication methods, this extended ARC can simultaneously support both TCP and FAST with little overhead. In order to make ARC support a distributed utility control without an explicit utility control at network routers, an extra step is required to modify the protocol to adapt a compatible network-evaluated link price instead of the source-estimated link price. The conversion issues are not discussed further, since it is out of scope of this paper. Also, we leave stability analysis and evaluation of the modified FAST system as future work.

VI. Conclusion

In this paper, we present Aggregate Rate Controller (ARC), a reduced parameter proportional-integral controller for Internet traffic. ARC resolves configuration difficulties that have limited deployment of past AQM approaches by taking a stable and efficient proportional-integral controller design for AQM, carefully reducing the control parameters based on a sound understanding of Internet congestion control, and providing practical configuration guidelines though control engineering for resilient performance over a wide range of traffic conditions. Also, by using a low frequency, direct rate-based measurement of traffic load rather than an indirect queue-based measurement, ARC significantly reduces control noise while providing flexible QoS to fulfill the needs of various Internet applications. Thus, ARC eases the difficulties in the configuration of proportional-integral control for PI [3] and REM [6], optimizes performance through an efficient rate-based method for detecting congestion, and, with configuration guidelines that match network operator knowledge, is a step towards practical deployment.

Our simulations demonstrate that by complying with the configuration guidelines, ARC can efficiently support a wide-range of traffic conditions, dampening queue oscillations, keeping average queues low and throughput high, even when the configuration is not optimized for the current traffic. ARC out-performs PI [3], AVQ [4] and SFC [2] when taken over all tested conditions in terms of queue dynamics, throughput, data loss rate and Web object service time. PI is shown to have underutilization for lightly loaded conditions, the norm for many Internet routers, as well as underutilization for sudden traffic load changes, owning to PI's high frequency queue-based controller design. AVQ is shown to have poor control of queue oscillations in lightly loaded conditions owning to AVQ's virtual queue marking mechanism, and is sensitive to control parameter settings. SFC is shown to have queue lengths that are a function of the traffic load, owning to SFC's lack of integral control, making it ineffective over a wide range of load conditions.

In addition to TCP support, ARC can be easily extended to support other similar distributed utility/congestion control protocols with little overhead, since most of the protocols use queuing delay-based link utility price estimation methods that are fundamentally compatible with one another. As an example, we extend ARC to concurrently support TCP and FAST [14], an IP compatible distributed utility control protocol proposed to overcome the inefficiency of AIMD for high capacity networks.

We leave stability analysis and evaluation of the FAST-extended ARC system as future work. Other future work includes extending ARC to dynamically adapt the controller parameters to the current traffic conditions. Such extensions would require additional stability and implementation analysis. In addition, extensive investigation of incremental deployment issues would be needed before adoption of ARC into the current Internet architecture.

REFERENCES

[1] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC-3168, September 2001.

[2] Yuan Gao and Jennifer Hou, "A State Feedback Control Approach to Stabilizing Queues for ECN-Enabled TCP Connections," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2003.

[3] 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*, 2001, pp. 1726–1734.

[4] Srisankar Kunniyur and R. Srikant, "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," in *Proceedings of ACM SIGCOMM*, San Diego, CA, USA, August 2001.

[5] G. Silva, A. Datta, and S. P. Bhattacharyya, "PI Stabilization of First-Order Systems with Time Delay," *Automatica*, December 2001.

[6] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, and Qinghe Yin, "REM: Active Queue Management," *IEEE Network*, vol. 15, no. 3, pp. 48–53, May/June 2001.

[7] C. V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong, "A Control Theoretic Analysis of RED," in *Proceedings of IEEE INFOCOM*, 2001, pp. 1510–1519.

[8] Katsuhiko Ogata, *Modern Control Engineering, Fourth Edition*, Prentice Hall, Upper Saddle River, New Jersey, USA, 2002.

[9] Sharad Jaiswal, Gianluca Iannaccone, Christophe Diot, Jim Kurose, and Don Towsley, "Inferring TCP Connection Characteristics Through Passive Measurements," in *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[10] Baek-Young Choi, Sue Moon, Zhi-Li Zhang, Konstantina Papagiannaki, and Christophe Diot, "Analysis of Point-To-Point Packet Delay in an Operational Network," in *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[11] Martin Arlitt and Tai Jin, "Workload Characterization of the 1998 World Cup Web Site," Tech. Rep. HPL-1999-35R1, HP Laboratories Palo Alto, September 1999.

[12] F. Hernandez-Campos, K. Jeffay, and F.D. Smith, "Tracing the Evolution of the Web Traffic: 1995-2003," in *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Orlando, FL, USA, Oct. 2003.

[13] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy, "An Analysis of Internet Content Delivery Systems," in *Usenix Operating Systems Design and Implementation (OSDI)*, Boston, MA, USA, Oct. 2002, pp. 315 – 327.

[14] F. Paganini, Z. Wang, S. H. Low, and J. C. Doyle, "A New TCP/AQM for Stable Operation in Fast Networks," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, April 2003.

[15] Dina Katabi, Mark Handley, and Charlie Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proceedings of ACM SIGCOMM*, Pittsburgh, PA, USA, August 2002.

[16] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer, "Equation-Based Congestion Control for Unicast Applications," in *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, August-September 2000, pp. 43–56.

[17] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communication*, pp. 1465–1480, October 1995.

[18] Jae Chung and Mark Claypool, "Analysis of Active Queue Management," in *Proceedings of The 2nd IEEE International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, April 2003.