Providing Fairness Between TCP NewReno and TCP Vegas with RD Network Services

Maxim Podlesny Carey Williamson
Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, AB, Canada T2N 1N4
{mpodlesn,carey}@ucalgary.ca

Abstract— **While Transmission Control Protocol (TCP) variants with delay-based congestion control (e.g., TCP Vegas) provide low queueing delay and low packet loss, the key problem with their deployment on the Internet is their relative performance when competing with traditional TCP variants with loss-based congestion control (e.g., TCP NewReno). In particular, the more aggressive loss-based flows tend to dominate link buffer usage and degrade the throughput of delay-based flows. In this paper, we study a novel approach for achieving fair sharing of the network resources among TCP variants, using Rate-Delay (RD) Network Services. In particular, loss-based and delay-based flows are isolated from each other and served via different queues. Using extensive ns-2 network simulation experiments, we show that our approach is effective in providing fairness between loss-based NewReno and delay-based Vegas flows.**

I. Introduction

The Transmission Control Protocol (TCP) has been the dominant transport-layer protocol on the Internet for many years. TCP provides reliable byte stream delivery between end-host applications, using a suite of mechanisms for connection management, flow control, and error control. A congestion control mechanism was also added to TCP in 1988 [1].

There are two different paradigms for TCP congestion control. In the *loss-based* approach, a TCP source keeps increasing its sending rate until a bottleneck link buffer is saturated, and a data packet is dropped. Such a packet drop provides an implicit congestion signal for the TCP source to decrease its sending rate. In the *delay-based* approach, a TCP source carefully monitors the observed Round Trip Time (RTT) on the network, in order to detect the onset of congestion. In particular, a sudden increase in the end-to-end delay is an implicit congestion signal. With luck, congestion can be detected sooner, before packet losses occur. Assuming suitable measurement accuracy at TCP sources, the delay-based approach can provide high link utilization, with low queueing delay and near zero packet loss.

The main challenge with the deployment of delay-based protocols is their relative throughput disadvantage when competing with loss-based protocols [2]. In particular, the presence of aggressive loss-based flows can completely fill queue buffers at network links, and significantly deteriorate the throughput of delay-based flows. For example, Figure 1 shows how a single TCP NewReno flow and a single TCP Vegas flow share a 2 Mbps bottleneck link in a 60-second ns-2 network simulation experiment. In this example, the NewReno flow consumes approximately 92% of the bottleneck link bandwidth in steady-state, while the Vegas flow receives very little.

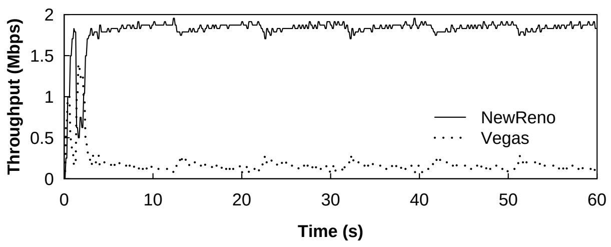


Fig. 1. Example of unfairness between TCP NewReno and TCP Vegas

The issue of compatibility between loss-based and delay-based protocols has triggered substantial research over the past decade. Many proposed solutions [3]–[5] rely on an end-to-end approach, which assumes the deployment of the proposed algorithms on end hosts without any help from the network. While these techniques can be effective, there are practical obstacles to their widespread deployment. Furthermore, the performance achieved can be highly sensitive to the configuration of their parameters.

In this paper, we explore a network-based solution to provide compatibility between loss-based and delay-based transport protocols. The main idea of such a solution is to isolate loss-based flows from delay-based flows, and service them via different queues at network links. As the basis for our approach, we use Rate-Delay (RD) Network Services [6], which were originally proposed for serving delay-sensitive and throughput-oriented applications. We apply RD services to TCP flow isolation, using TCP NewReno [7] and TCP Vegas [8], [9] as representatives of loss-based and delay-based protocols, respectively. Through extensive simulations, we demonstrate that isolating these types of flows is effective. In particular, the protocols share the network resources fairly for a wide variety of network topologies and traffic scenarios considered. In addition to the performance advantages of our approach, there are also practical advantages. First, our solution does not require any changes in the end-to-end functionality of the transport protocols. Second, our approach is simple to configure, and very robust in its performance. Furthermore, this approach is applicable to any loss-based or delay-based protocol.

The rest of this paper is structured as follows. Section II provides relevant background material regarding TCP, TCP variants, and RD Network Services. Section III presents the design details of our proposed solution. Section IV presents simulation experiments evaluating the performance and robustness of our solution. Section V discusses practical issues, while Section VI reviews prior related work. Finally, Section VII concludes the paper.

II. Background

In this section, we briefly describe TCP NewReno, TCP Vegas, and RD Network Services.

A. TCP NewReno

TCP congestion control relies on the dynamic manipulation of the window size used by TCP's window-based flow control. In particular, a TCP source controls its average data sending rate by adjusting its congestion window size, which is an estimate of how much data can be safely transmitted into the network without experiencing packet loss.

To determine an appropriate data sending rate, the TCP source operates in two different modes: *slow start* and *congestion avoidance*.

1) Slow start: A NewReno source begins transmitting a new data flow in slow start. The initial congestion window is typically one segment. Each time a NewReno source receives an acknowledgment (ACK) packet, it increases the congestion window cwnd by one segment. With this approach, the congestion window size expands multiplicatively, doubling every RTT. An additional TCP parameter, the slow start threshold, determines when this aggressive window expansion mode should end. Once the threshold is reached, a NewReno flow transitions into congestion avoidance.

2) Congestion avoidance: In congestion avoidance, a NewReno source increases its congestion window linearly, rather than multiplicatively. Upon receiving each ACK, the congestion window is changed as follows:

$$cwnd = cwnd + \frac{1}{cwnd} \tag{1}$$

Thus, each subsequent successful delivery of a data packet leads to a small increase of TCP congestion window, and after the exchange of a complete window's worth of data, cwnd increases by one segment. This linear increase continues until the flow is finished, or a packet loss occurs.

3) Loss recovery: There are two ways for a NewReno source to detect packet losses. One approach is a *coarse-grained timeout*, which happens when a NewReno source has not received any ACK during a Retransmission Timeout (RTO) interval. When a timeout occurs, a NewReno source resets the congestion window size to one packet, and resumes slow start. Another approach is called *triple duplicate ACK*, in which three repeated cumulative ACKs carry the same redundant information, indicating a lost packet within a window of packets. In this case, a NewReno source decreases the congestion window size by half.

B. TCP Vegas

TCP Vegas is one of the most prominent examples of delay-based transport protocols. The key difference between loss-based protocols like TCP NewReno and delay-based protocols like TCP Vegas is that the latter uses changes in end-to-end delay (rather than loss) as the means for detecting congestion. In particular, since TCP throughput is inversely related to RTT, Vegas measures the difference between the expected throughput and the actual throughput. The idea is that the actual throughput should match the expected throughput if there is no congestion along the network path. A lower actual throughput indicates increased delay, and hence congestion, on the network path.

Similar to NewReno, Vegas has slow start and congestion avoidance modes.

1) Slow start: One difference between NewReno and Vegas is that the initial congestion window size is two packets instead of one. Another difference is that Vegas doubles its congestion window every other RTT, rather then every RTT. This approach improves measurement accuracy, since the congestion window is fixed when comparing $ExpectedThruput$ and $ActualThruput$. When the value of $diff$ in Equation (2) exceeds a threshold parameter γ, Vegas switches to congestion avoidance mode.

2) Congestion avoidance: The adjustments of the congestion window size are made based on the value of $diff$, which determines both the direction and the magnitude of adjustment required. The value of $diff$ is given by:

$$diff = (ExpectedThruput - ActualThruput) * BaseRTT \tag{2}$$

where $ExpectedThruput$ is the expected throughput, $ActualThruput$ is the actual observed throughput, and $BaseRTT$ is the minimum RTT observed for the network path. The $ExpectedThruput$ and $ActualThruput$ are calculated using:

$$ExpectedThruput = \frac{cwnd}{BaseRTT} \tag{3}$$

$$ActualThruput = \frac{cwnd}{RTT} \tag{4}$$

where $cwnd$ is the current congestion window size (in segments), and RTT is the actual RTT observed. A Vegas source updates its congestion window size once per RTT, as follows:

$$cwnd = \begin{cases} cwnd + 1 & if\ diff\ < \ \alpha \\ cwnd - 1 & if\ diff\ > \ \beta \\ cwnd & otherwise \end{cases}$$

where α and β are specified parameters. In other words, Vegas increases $cwnd$ by one packet if the per-flow queue at a bottleneck link is smaller than α, decreases $cwnd$ by one packet if the per-flow queue is larger than β, and keeps $cwnd$ unchanged otherwise. In our work, we use the typical settings of these parameters, namely $\alpha = 1$ and $\beta = 3$ [2].

3) Loss recovery: The loss signals for TCP Vegas are similar to those for TCP NewReno. One difference is that the congestion window after a timeout is 2 packets. Furthermore, if a loss is detected with triple duplicate ACKs, then the congestion window decreases by 25%, rather than 50%.

C. RD Network Services

The key idea in Rate-Delay (RD) Network Services is to divide the incoming traffic into two classes, and service them through two First-In First-Out (FIFO) queues. The R queue serves throughput-greedy applications, while the D queue serves delay-sensitive applications. Network applications are responsible for indicating their requirements in the Type of Service (ToS) field of the Internet Protocol (IP) datagram header [10].

The scheduling algorithm in RD Network Services ensures that the relative traffic volumes dispatched from the R and D queues obey:

$$\lambda = \frac{n_D}{kn_R} \tag{5}$$

where n_R and n_D are the numbers of active flows from classes R and D (respectively) passing through the link, and k is a configurable parameter to indicate the desired target ratio for the average per-flow throughputs for classes R and D.

Mathematically, the service rates for the R and D queues are given by:

$$S_R = \frac{kn_R C}{n_D + kn_R} \tag{6}$$

$$S_D = \frac{n_D C}{n_D + kn_R} \tag{7}$$

where C is the link capacity. The number of flows in each class is estimated using the time-stamp vector algorithm [11], [12]. The buffer size B_D for the D queue is configured so that the maximum queueing delay of D packets does not exceed a specified bound d; the rest of the buffer space B_R is allocated for the R queue. Thus, the buffer size of the R queue is large enough to provide full utilization of the link capacity dedicated for throughput-greedy applications. In the event of queue overflow, the DropTail policy applies for both queues.

To accommodate the dynamic nature of Internet traffic flows, each RD link periodically recalculates the main control parameters S_R, S_D, B_R, and B_D. The details of the architecture are described elsewhere [6]. In the next section, we describe how we apply RD Network Services to handle loss-based and delay-based transport protocols.

III. Solution Details

We serve NewReno traffic through the R queue, and Vegas traffic through the D queue, using the scheduling algorithm presented in Figure 2. It is assumed that each TCP sender performs appropriate packet marking for RD Network Services. To estimate the number of active flows, we use the same timestamp vector algorithm as in [6].

In the link scheduling, the selection of a queue to be served is based on the values L_R and L_D, which indicate the traffic volume served from the R and D queues, respectively, since

```
/* select the queue to transmit from */
**if** $q_R > 0$ **and** $q_D > 0$ **then**
    **if** $kn_R L_D > n_D L_R$ **then**
        $x \leftarrow$ R;
    **else**
        $x \leftarrow$ D;
**else** /* exactly one of the R and D buffers is empty */
    $x \leftarrow$ class of the non-empty buffer;
$p \leftarrow$ first packet in the $x$ queue;
$s \leftarrow$ size of $p$;
**if** $p$ != **null then**
    /* update the $L$ variables */
    **if** $q_R > 0$ **and** $q_D > 0$ **then**
        $L_x \leftarrow L_x + s$;
        $\delta L \leftarrow \frac{L_R n_D}{kn_R} - L_D$;
        **if** $\delta L < 0$ **then** $\delta L \leftarrow 0$;
    **else** /* only D buffer is empty */
    **if** $q_R > 0$ **and** $q_D = 0$ **then**
        $L_R \leftarrow 0$; $L_D \leftarrow 0$;
    **else** /* only R buffer is empty */;
        **if** $\delta L > 0$ **then** $\delta L \leftarrow \delta L - s$;
        **if** $\delta L > 0$ **then**
            $L_D \leftarrow -\delta L$;
        **else**
            $L_D \leftarrow 0$;
        $L_R \leftarrow 0$;
    transmit $p$ into the link;
    $q_x \leftarrow q_x - s$
```

Fig. 2. RD router operation when link is idle, and buffer is non-empty.

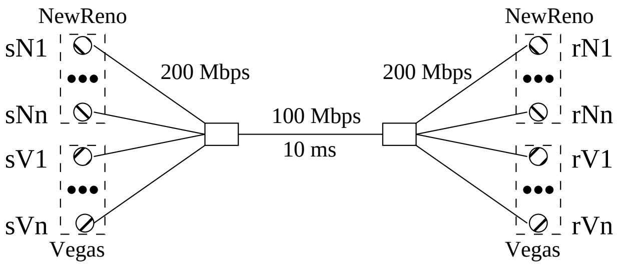


Fig. 3. Simple dumbbell network topology

the start of the most recent recalculation interval. If L_D does not exceed λL_R, then the D queue is chosen for transmission. Otherwise, the next transmission is from the R queue.

Other relevant parameters are configured as follows. We set $k = 1$, since our goal is fair sharing between TCP variants at the bottleneck link. For the buffer size, we allocate 10% of the total buffer space to the D queue, and the remainder to the R queue. Determining the optimal buffer allocation will be explored in future work.

IV. Simulation Evaluation and Results

In this section, we use version 2.29 of the ns-2 network simulator [13] to evaluate the performance of concurrent TCP

NewReno and TCP Vegas flows sharing a bottleneck link. All flows use packets of size 1 KB. Each experiment lasts 60 s, with 5 replications for each of the considered parameter settings. The performance metrics for average link utilization and average packet loss are calculated using all but the first 5 seconds (warmup) of the experiment. For calculating instantaneous link utilization and packet loss, we calculate and plot the results every 0.2 s interval.

The primary performance metrics are the average utilizations achieved by NewReno flows and by Vegas flows, and the average packet loss rate of Vegas flows. As a secondary metric, we use a *relative fairness index* to quantify the compatibility between NewReno and Vegas flows. This metric is defined as the ratio of the average per-flow utilizations for Vegas and NewReno in each experimental setting. A value near 1 indicates fairness between NewReno and Vegas flows.

In most of our experiments, we use a simple dumbbell network topology, as shown in Figure 3. In this topology, the access links have a capacity of 200 Mbps, and the core bottleneck link capacity is 100 Mbps. The propagation delay of the bottleneck link is 10 ms. We configure the link buffers to be proportional to the delay-bandwidth product of the network. Specifically, the buffer sizes are set to $C \cdot 250$ ms, where C is the capacity of the link.

The traffic scenarios vary from one experiment to the next. In most scenarios, there are 100 long-lived TCP NewReno flows and 100 long-lived TCP Vegas flows in each of the forward and reverse directions. The RTTs of the flows are chosen uniformly at random from the interval between 20 ms and 300 ms.

A. Effect of Number of NewReno Flows

In the first simulation experiment, we study the scalability of our solution with respect to the number of NewReno flows. In this scenario, we fix the number of long-lived Vegas flows (100) and vary the number of long-lived NewReno flows from 10 to 700. The number of flows in the reverse direction is the same as in the forward direction for both types of flows.

Figure 4 shows the simulation results from this experiment. The results are plotted with respect to the NewReno *flow ratio* (i.e., the ratio of NewReno flows to Vegas flows), which varies from 0.1 to 7.0.

Figure 4(a) shows that our solution provides proper sharing of the network between the two flow types, in proportion to their prevalence. When the flow ratio is 1, both flow types share the network equally. When there are many NewReno flows, their aggregate utilization is proportional to their prevalence on the network.

The relative fairness index is close to 1 across the range of settings studied. For example, when the NewReno flow ratio is 3 (i.e., 300 NewReno flows and 100 Vegas flows), then the utilizations of NewReno and Vegas flows are 74% and 24%, respectively. The fairness index is $0.24 \cdot \frac{0.74}{3} = 0.97$.

Figure 4(b) shows that the packet loss rate for Vegas grows linearly with the number of NewReno flows. This happens because the service rate of the D queue decreases.

B. Effect of Number of Vegas Flows

The second experiment investigates the scalability of our solution with respect to the number of Vegas flows. We fix the number of long-lived NewReno flows (100), and vary the number of long-lived Vegas flows from 10 to 700. The Vegas flow ratio (i.e., the ratio of Vegas flows to NewReno flows) varies from 0.1 to 7.0.

Figure 5(a) again shows that our approach achieves the desired sharing of network resources. As Vegas flows become prevalent, they obtain a larger share of the network bandwidth. The relative fairness index is close to 1 for all settings considered.

Figure 5(b) shows that the packet loss rate for Vegas increases non-linearly, unlike the previous scenario. This difference can be explained as follows. In the previous scenario (fixed Vegas flows and varied NewReno flows), the loss rate of Vegas depends on the (varied) service rate of the D queue. In the current scenario (fixed NewReno flows and varied Vegas flows), the loss rate is affected not only by the (varied) service rate of the D queue, but also by the (fixed) buffer space for Vegas flows. The packet loss rate increases because of the increased competition for the finite Vegas buffers.

C. Effect of Web-like Traffic Flows

The previous experiments considered only long-lived TCP flows, and their steady-state performance. In this scenario, we augment our baseline scenario with Web-like traffic flows. Their presence dynamically varies the number of active TCP flows, and the competition for link buffer space, providing a more realistic model of Internet traffic dynamics.

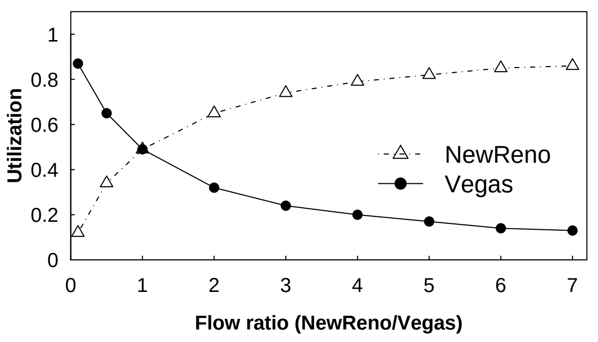
For this simulation experiment, we added two Web servers (one for each TCP variant) to our simulation model, and connected them to the bottleneck link. We varied the intensity of Web-like traffic between 1 flow per second (fps) and 300 fps. The Web-like flows arrive according to a Poisson process. The size of Web-like flows is Pareto distributed with a mean of 30 KB and a shape index 1.3. The propagation RTT for each flow was 100 ms.

Figure 6 shows the results from this simulation experiment. The results show that our solution is quite robust to the dynamics of Web-like traffic. In particular, the relative fairness index ranges between 0.75 and 0.96 across the scenarios considered. The packet loss rate of Vegas flows increases with the intensity of Web-like traffic. This result is as expected, because of the increased burstiness of traffic.

D. Effect of Link Capacity

We next explore the capacity scalability of our flow isolation approach, by varying the bottleneck link capacity between 10 Mbps and 1 Gbps. As in Section IV-C, we consider Web-like traffic flows, but with a fixed average intensity of 50 fps.

The results in Figure 7 show fairly robust performance for our solution, except at very low link speeds. In particular, the relative fairness index is between 0.72 and 0.88 for all scenarios except that for 10 Mpbs. For that capacity value, the throughput of TCP Vegas drops due to the small buffer

(a) 

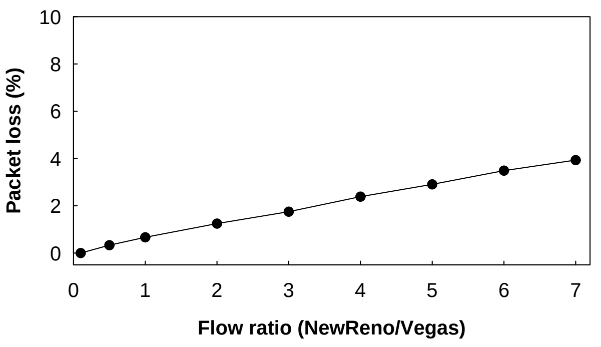
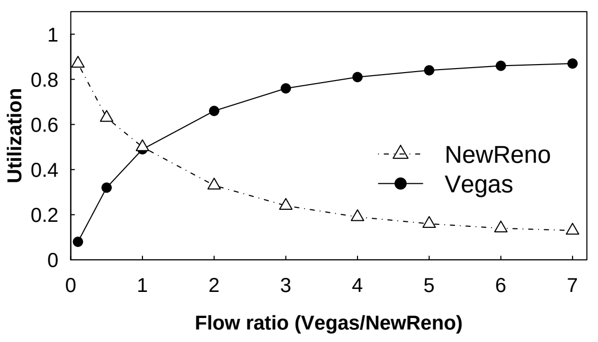
(b) 

Fig. 4. Dependence on the number of TCP NewReno flows: (a) average utilizations of TCP NewReno and TCP Vegas; (b) average loss rate of TCP Vegas.

(a) 

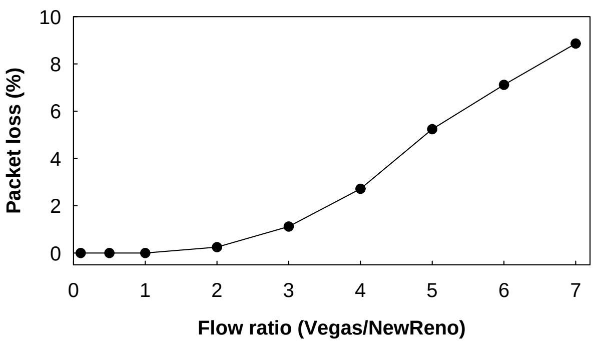
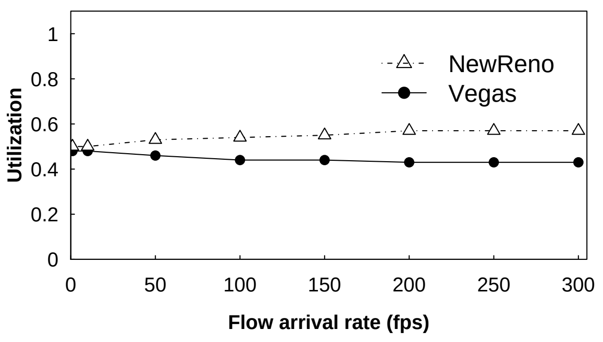
(b) 

Fig. 5. Dependence on the number of TCP Vegas flows: (a) average utilizations of TCP NewReno and TCP Vegas; (b) average loss rate of TCP Vegas.

(a) 

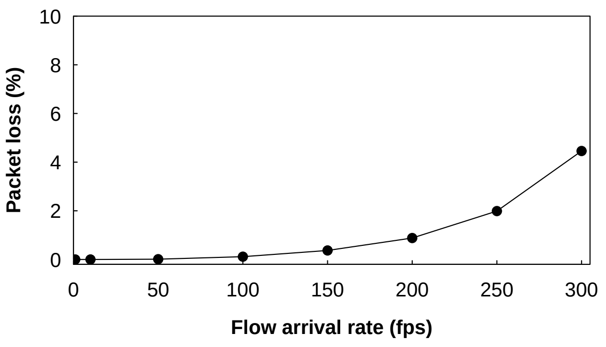
(b) 

Fig. 6. Influence of the intensity of Web-like traffic: (a) average utilizations of TCP NewReno and TCP Vegas; (b) average loss rate of TCP Vegas.

size (31.25 KB), and the fairness index is 0.61. The packet loss rate for TCP Vegas is absurdly high (28%) at this buffer size, but decreases quickly as the bottleneck link capacity (and buffer size) is increased.

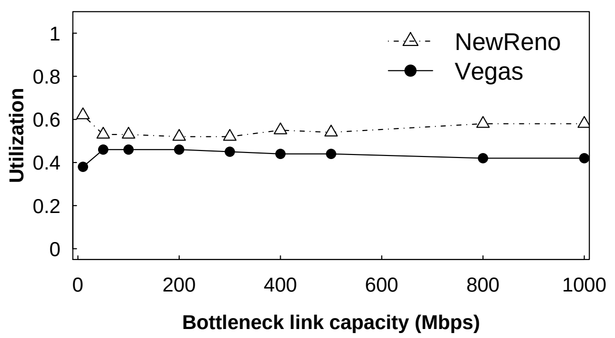
E. Effect of RTT

We study the influence of propagation RTT by fixing the minimum propagation RTT for long-lived flows at 20 ms, and varying the maximum propagation RTT from 30 ms to 1 s. Flow propagation RTTs are chosen uniformly at random in the range between the minimum RTT and the maximum RTT. The traffic load is the same as in Section IV-D.

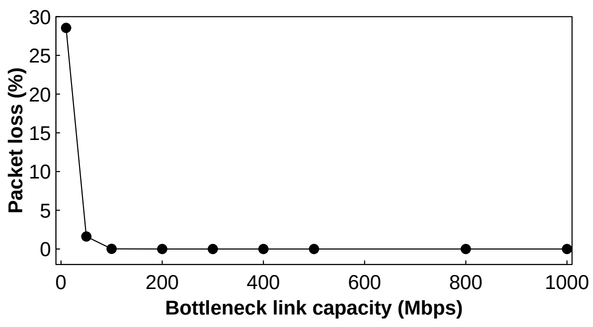
Figure 8 shows very robust performance for our solution across the range of RTT values considered. The relative fairness index ranges between 0.83 and 0.87. The packet loss rate is well below 1% for all parameter settings considered. However, its behavior is non-monotonic: it decreases from 0.29% for 20 ms to 0.01% for 200 ms, and then increases to 0.11% for 1000 ms. The wide variation in flow RTT values produces high variability in the packet queueing and loss behaviour.

F. Effect of Traffic Burstiness

In all previous simulation experiments, we have assumed that the flow arrival process is Poisson. To explore how traffic burstiness affects the performance of protocol-based isolation, we use Web-like flows arriving according to Pareto distribution with shape index 1.1. This model approximates the self-

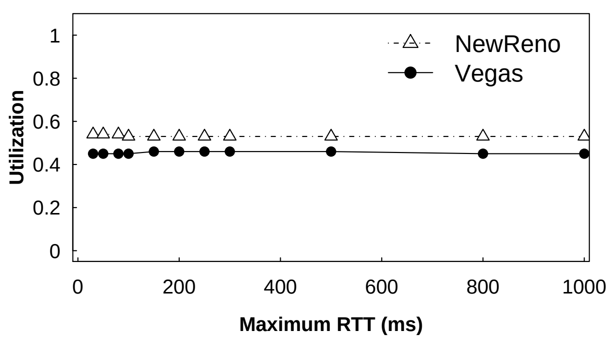


(a)

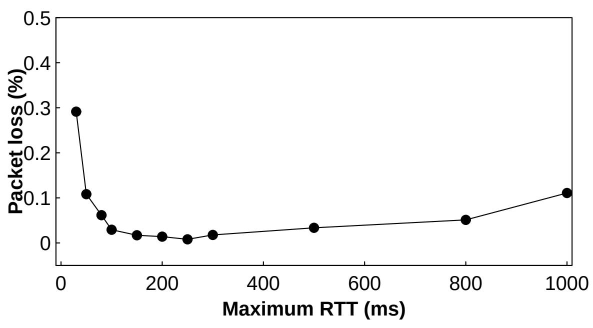


(b)

Fig. 7. Capacity scalability: (a) average utilizations of TCP NewReno and TCP Vegas; (b) average loss rate of TCP Vegas.



(a)



(b)

Fig. 8. Influence of propagation RTT: (a) average utilizations of TCP NewReno and TCP Vegas; (b) average loss rate of TCP Vegas.

similarity observed in Internet traffic [14].

We run the experiments for Web-like traffic intensities of 50 fps, 100 fps, and 200 fps. The sizes of Web-like flows are Pareto distributed with shape index 1.3, and we set the average flow size L_{avg} first to 30 KB and then to 100 KB. The long-lived flows are the same as in Section IV-E.

Figure 9 reports the throughputs and loss rates for NewReno and Vegas. The relative fairness index is 0.9 for 50 fps, but decreases to 0.69 for 200 fps when L_{avg} = 30 KB. For L_{avg} = 100 KB, the relative fairness index is 0.86 for 50 fps, but decreases to 0.67 for 200 fps. The decline in fairness can be explained by the increase in the packet loss rate for Vegas.

G. Effect of Sudden Traffic Changes

The next simulation experiment studies the behavior of our solution in the presence of sudden changes in the numbers of NewReno and Vegas flows, due to flash crowds or correlated arrivals. In this simulation only, the long-lived flows are unidirectional. The experiment lasts for 180 s.

Figure 10(a) shows the dynamics of the traffic flows considered. For example, during the interval [60 s; 80 s] there are 20 Vegas flows and 100 NewReno flows in the network. At time 80 s, the number of Vegas flows increases suddenly to 100, while at time 100 s, the number of NewReno flows drops to 0. These dynamics create sudden changes in the composition of traffic at the bottleneck link.

We run this simulation experiment to evaluate the performance of our scheme compared to traditional DropTail queues as well as Random Early Detection (RED) with Explicit Congestion Notification (ECN) [15], [16]. Figure 10(b) shows the results for NewReno and Vegas flows with DropTail queueing. These results indicate a sluggish response to sudden changes in the flow mix, and a struggle to achieve fairness between TCP variants. Figure 10(c) shows the results for RED/ECN. While the behaviour has improved compared to DropTail, response is still sluggish, and NewReno still shows distinct throughput advantages over Vegas (e.g., see the results from 40 s to 60 s, and from 80 s to 100 s). Figure 10(d) shows the results for our solution using RD Network Services. The transient response of our algorithm is immediate, and the fairness objective is achieved throughout. Figure 10(e) compares the relative fairness of each approach. The relative fairness index is approximately unity with our solution. While the behaviour of RED/ECN in Figure 10(c) resembles that of our solution, RED/ECN still has inferior TCP fairness. The performance of DropTail is even worse.

H. Multi-bottleneck Topology

Our final simulation experiment considers a multi-bottleneck topology. In particular, we use a parking-lot topology as shown in Figure 11. This topology has access links of capacity 200 Mbps, and five consecutive bottleneck links, each of capacity 100 Mbps. Each bottleneck link has a propagation delay of 5 ms.

The traffic flows on this topology are defined as follows. Each of the six pools A,B,C,D,E, and F generates 20

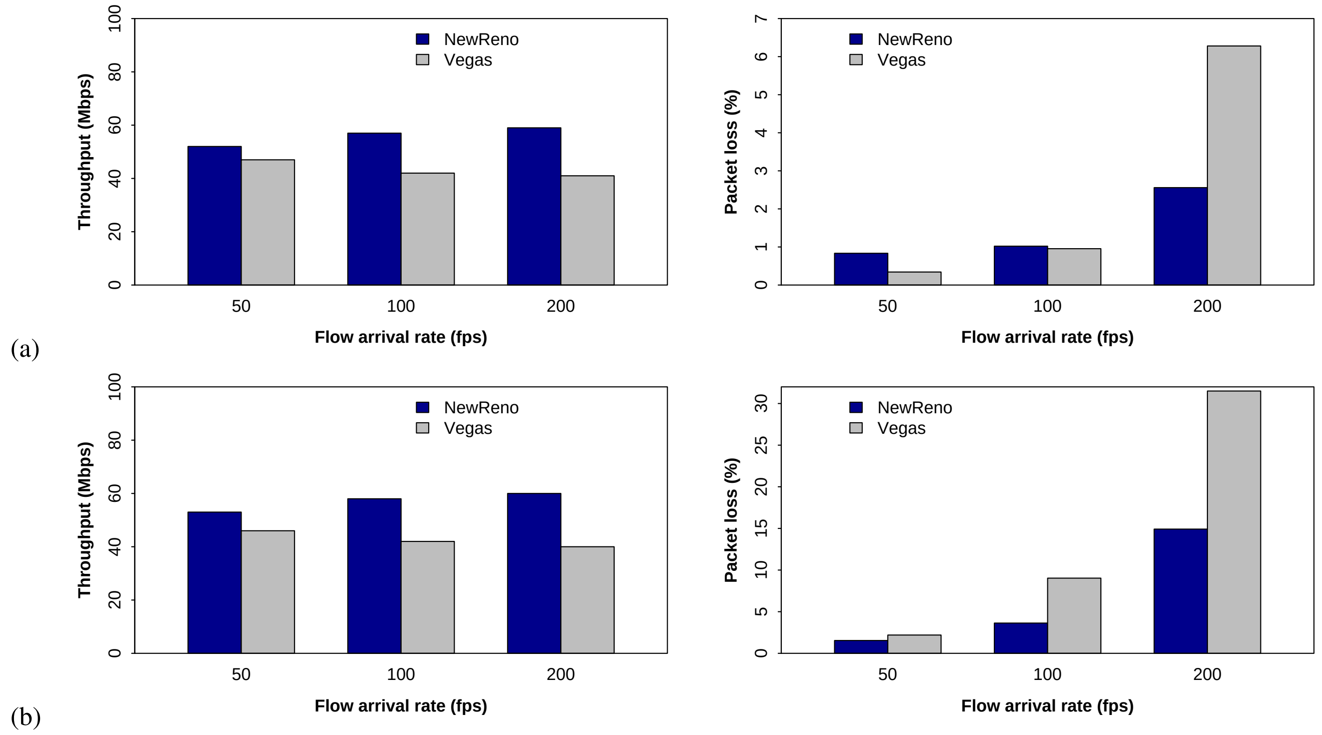


Fig. 9. Heavy-tailed arrival of Web-like traffic with its different average flow size L_{avg}. Average throughputs of TCP NewReno and TCP Vegas, and average loss rates of TCP NewReno and TCP Vegas: (a) L_{avg} = 30 KB; (b) L_{avg} = 100 KB.

NewReno and 20 Vegas long-lived flows. The flows starting in pool A traverse the entire network and arrive at destinations in pool H. The flows starting in pool B are destined to pool C, and thus traverse only one bottleneck link. The same applies for flows starting in pools C,D,E, and F, which have destinations in pools D,E,F, and G, respectively (i.e., they traverse only one bottleneck link). Thus, each bottleneck link handles 40 NewReno and 40 Vegas flows. There are also 20 NewReno and 20 Vegas long-lived flows going in the reverse direction, from pool H to pool A. The propagation RTTs of all flows are distributed uniformly at random between 60 ms and 300 ms.

As with the previous simulation experiment, we compare results for DropTail, RED/ECN, and our flow isolation scheme. Figure 12 shows the throughputs of NewReno and Vegas flows. From left to right, the results correspond to the five bottleneck links. For each link, there are three policies (DropTail, RED/ECN, and RD). Within each policy, there are vertical bars showing the relative throughputs of NewReno and Vegas flows, in that order. Tall bars of equal height are desired.

The results in Figure 12 show that NewReno and Vegas achieve approximately the same throughput at each bottleneck link with our scheme. On the other hand, NewReno flows have a substantial advantage in DropTail and in RED/ECN. While RED/ECN improves the relative fairness compared to DropTail, the fairness index is still significantly less than 1.

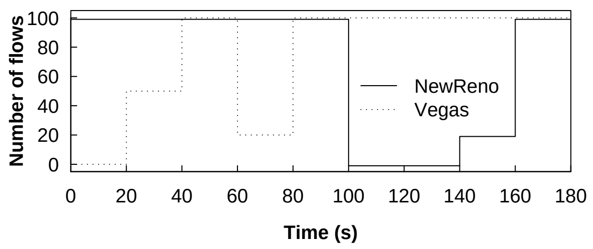
V. Discussion

The evaluation of our approach has the premise that each TCP sender performs appropriate marking of its packets at the source. Inappropriate classification of packets by senders is an issue that warrants discussion. We can distinguish two potential scenarios that may arise: Vegas senders masquerading as NewReno, and NewReno senders masquerading as Vegas.

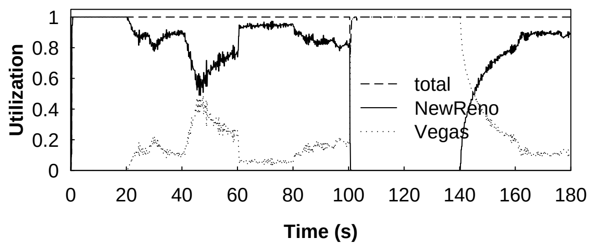
In the first scenario, a Vegas sender marks its packets as NewReno. However, this (mis)behavior provides no advantage for a (misbehaving) Vegas sender, since it leads to the same problem of compatibility of NewReno and Vegas flows served through the same DropTail link. Therefore, there is no reason for a Vegas sender to misbehave in this way.

The second scenario takes place when a NewReno sender identifies its packets as Vegas. This could provide a throughput advantage for the misbehaving flow, and lead to serious throughput degradation for Vegas flows. To gain a performance advantage, a misbehaving NewReno flow needs at least as much buffer space as it obtains when marking packets properly. However, many NewReno flows sharing the same bottleneck link can potentially misbehave the same way, and a misbehaving NewReno flow does not have information about this quantity. In addition, the buffer size of a queue serving Vegas traffic is much smaller than for a queue serving NewReno traffic. Therefore, inappropriate marking of packets by a NewReno flow will not improve its performance. Moreover, the danger for a misbehaving NewReno flow from such a behavior is that it could degrade its own throughput.

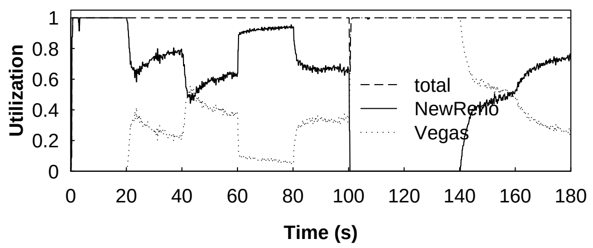
As future work, we plan to explore the question of how



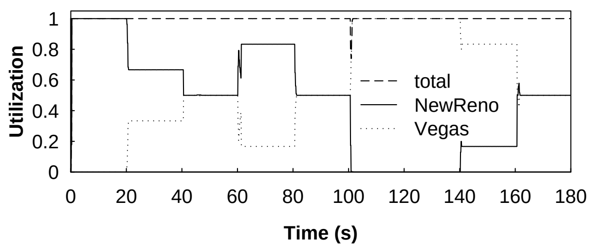
(a) NewReno and Vegas flows



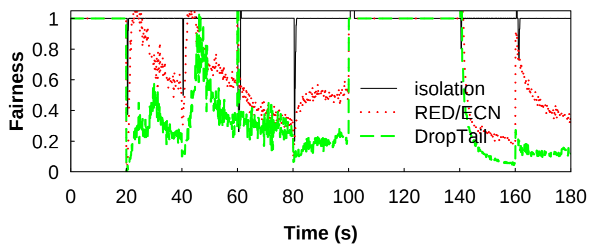
(b) DropTail



(c) RED/ECN



(d) Our solution



(e) Relative Fairness

Fig. 10. Influence of sudden changes in network traffic flows

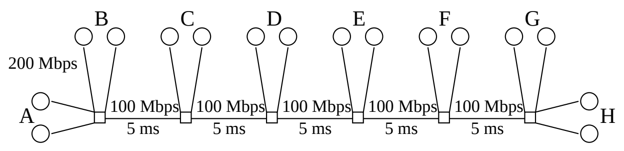


Fig. 11. Parking-lot topology

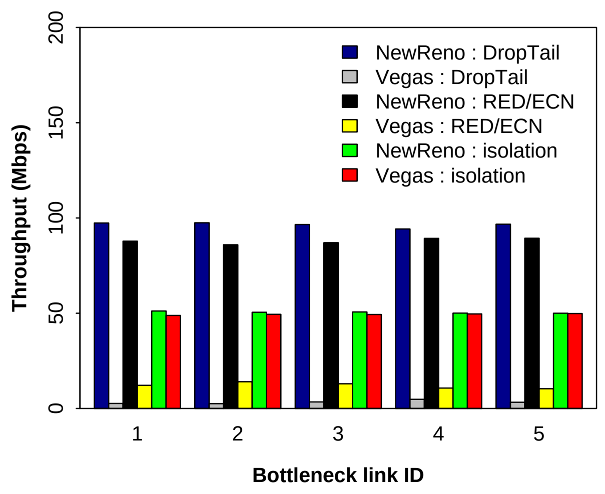


Fig. 12. Multi-bottleneck link topology. The average throughputs of TCP NewReno and TCP Vegas for different link serving schemes.

to minimize risk due to NewReno misbehavior. One potential solution could exploit the packet-level traffic characteristics. In particular, Vegas flows are inherently smoother, since the congestion window size adjustments are more gradual.

VI. Related Work

Delay-based congestion control has been widely studied in the literature [17]–[23]. In [24], the authors propose a new transport protocol, TCP Illinois, which combines both loss-based and delay-based approaches. In particular, information about losses is used to make decisions regarding whether the congestion window should be increased or decreased, while information about queueing delay is used to determine the magnitude of the adjustment. FAST TCP [25], [26] also relies on both losses and queueing delay as congestion signals. Boyden et al. [27] advocated using TCP Vegas as a transport protocol for streaming media applications. Their results show that using TCP Vegas is feasible under a wide range of network conditions. Tang et al. [28] studied equilibrium in general multi-protocol networks. Kuzmanovic and Knightly [29] designed TCP Low Priority (TCP-LP), which allows low-priority applications such as bulk data transfer to use excess bandwidth without disturbing best-effort service such as TCP NewReno. TCP-LP uses end-to-end delay to estimate the available bandwidth.

In [5], the authors proposed two approaches for improving fairness between TCP Vegas and TCP Reno. The first approach modifies the original TCP Vegas; the resulting new version, TCP Vegas+, is competitive against TCP Reno. The second approach, ZL-RED, considers TCP Vegas and TCP Reno flows as well-behaving and misbehaving, respectively, and drops more packets from misbehaving flows. The main problem with the proposed solutions is that they are highly sensitive to configuration parameters. Vendictis and Baiocchi [30] proposed a model for calculating the throughputs of one TCP Reno and one TCP Vegas source sharing the same link.

Feng and Vanichpun [31] showed that the default configuration of TCP Vegas, i.e., $\alpha = 1$ and $\beta = 3$, was incompatible with TCP Reno. The authors proposed two approaches for configuring α and β, in which the configuration parameters are defined by the buffer size and the numbers of Reno and Vegas flows. However, the delivery of such information requires a change in the current packet structure, and increases the feedback overhead.

Kotla and Reddy [4] proposed mPERT, a modification of PERT (Probabilistic Early Response TCP) [32]. mPERT tries to share bandwidth fairly with loss-based protocols through an appropriate adjustment of the sending rate. If the delay exceeds 50% of the maximum queueing delay observed, then mPERT assumes that there are loss-based flows present. Budzisz et al. [3] proposed a new back-off policy for achieving fairness between loss-based and delay-based flows. Specifically, a delay-based flow would operate in loss-based mode when sharing the network with loss-based flows, and would operate in delay-based mode otherwise. While both of these schemes show good performance, they require careful setting of the parameters.

VII. Conclusion

In this paper, we proposed network-level isolation of loss-based and delay-based transport protocols as a solution to the problem of TCP compatibility. In particular, our approach serves loss-based and delay-based traffic through different link queues.

Through extensive simulations, we have demonstrated that isolation of flow classes provides fair usage of the network resources by TCP NewReno and TCP Vegas. There are also significant deployment advantages to our scheme, since it can be deployed without requiring per-flow state [33], and requires no changes in the TCP packet structure or end-to-end functionality. Furthermore, our approach has relatively few configuration parameters (k and B), and provides performance that is quite robust across a wide range of parameter settings.

In the future, we will investigate the problem of optimal buffer allocation for loss-based and delay-based protocols in more detail. Also, we will explore mechanisms for detecting loss-based flows sharing the same link queue with delay-based flows.

References

[1] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings ACM SIGCOMM 1988*, August 1988.

[2] J. Mo, R. La, V. Anantharam, and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas," in *Proceedings IEEE INFOCOM 1999*, March 1999.

[3] L. Budzisz, R. Stanojevic, A. Schlote, R. Shorten, and F. Baker, "On the Fair Coexistence of Loss- and Delay-based TCP," in *Proceedings IEEE IWQoS 2009*, July 2009.

[4] K. Kotla and N. Reddy, "Making a Delay-based Protocol Adaptive to Heterogeneous Environments," in *Proceedings IEEE IWQoS 2008*, June 2008.

[5] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and Improvement of Fairness between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet," in *Proceedings IEEE ICNP 2000*, November 2000.

[6] M. Podlesny and S. Gorinsky, "RD Network Services: Differentiation through Performance Incentives," in *Proceedings ACM SIGCOMM 2008*, August 2008.

[7] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 2582, April 1999.

[8] L. Brakmo, S. Malley, and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," in *Proceedings ACM SIGCOMM 1994*, August 1994.

[9] C. Samios and M. Vernon, "Modelling the Throughput of TCP Vegas," in *Proceedings ACM SIGMETRICS 2003*, June 2003.

[10] J. Postel, "Internet Protocol. DARPA Internet Program. Protocol Specification." IETF RFC 791, September 1981.

[11] H. Kim and D. O'Hallaron, "Counting Network Flows in Real Time," in *Proceedings IEEE GLOBECOM 2003*, December 2003.

[12] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms for Counting Active Flows on High Speed Links," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 925–937, October 2006.

[13] S. McCanne and S. Floyd, *ns Network Simulator. http://www.isi.edu/nsnam/ns/.*

[14] E. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, February 1994.

[15] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

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

[17] D. Leith, R. Shorten, G. McCullagh, J. Heffner, L. Dunn, and F. Baker, "Delay-based AIMD Congestion Control," in *Proceedings International Workshop on Protocols for FAST Long-Distance Networks (PDLFnet 2007*, February 2007.

[18] R. King, R. Baraniuk, and R. Riedi, "TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP," in *Proceedings IEEE INFOCOM 2005*, March 2005.

[19] R. Jain, "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *ACM Computer Communications Review*, vol. 19, no. 5, pp. 56–71, October 1989.

[20] J. Ahn, P. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: Emulation and Experiment," in *Proceedings ACM SIGCOMM 1995*, August 1995.

[21] P. Danzig, Z. Liu, and L. Yan, "An Evaluation of TCP Vegas by Live Emulation," in *Proceedings ACM SIGMETRICS 1995*, May 1995.

[22] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas Revisited," in *Proceedings IEEE INFOOM 2000*, March 2000.

[23] S. Low, L. Peterson, and L. Wang, "Understanding TCP Vegas: a Duality Model," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 226–235, June 2001.

[24] S. Liu, T. Basar, and R. Srikant, "TCP-Illinois: a Loss and Delay-based Congestion Control Algorithm for High-speed Networks," in *Proceedings International Conference on Performance Evaluation Methodolgies and Tools*, October 2006.

[25] C. Jin, D. Wei, and S. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," in *Proceedings IEEE INFOCOM 2004*, March 2004.

[26] J. Wang, D. Wei, and S. Low, "Modelling and Stability of FAST TCP," in *Proceedings IEEE INFOCOM 2005*, March 2004.

[27] S. Boyden, A. Mahanti, and C. Williamson, "TCP Vegas Performance with Streaming Media," in *Proceedings IEEE International Performance Computing and Communications Conference (IPCCC 2007)*, April 2007.

[28] A. Tang, J. Wang, S. Low, and M. Chiang, "Equilibrium of Heterogeneous Congestion Control: Existence and Uniqueness," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 824–837, August 2007.

[29] A. Kuzmanovic and E. Knightly, "TCP-LP: Low-Priority Service via End-Point Congestion Control ," *IEEE/ACM Transactions on Networking*, vol. 14, no. 4, pp. 683–696, August 2006.

[30] A. Vendictis and A. Baiocchi, "Modeling a Mixed TCP Vegas and TCP Reno Scenario," in *Proceedings IFIP Networking 2002*, May 2002.

[31] W. Feng and S. Vanichpun, "Enabling Compatibility Between TCP Reno and TCP Vegas," in *Proceedings IEEE Symposium on Applications and the Internet (SAINT 2003)*, January 2003.

[32] S. Bhandarkar, A. Reddy, Y. Zhang, and D. Loguinov, "Emulating AQM from End Hosts," in *Proceedings ACM SIGCOMM 2007*, August 2007.

[33] M. Podlesny and S. Gorinsky, "Stateless RD Network Services," in *Proceedings IFIP Networking 2010*, May 2010.