Accumulation-based Congestion Control

Yong Xia, David Harrison, Shivkumar Kalyanaraman,
Kishore Ramachandran, and Arvind Venkatesan

Abstract—**This paper[1] generalizes the TCP Vegas congestion avoidance mechanism and uses *accumulation*, buffered packets of a flow inside the network routers, as a congestion measure based on which a *family* of congestion control schemes can be derived. We call this model accumulation-based congestion control (ACC), which fits into the nonlinear optimization framework proposed by Kelly. The ACC model serves as a reference for packet-switching network implementations. We show that TCP Vegas is one possible scheme under this model. It is well known that Vegas suffers from round trip propagation delay estimation error and reverse path congestion. We therefore design a new Monaco scheme that solves these problems by employing an *out-of-band, receiver-based* accumulation estimator, with the support of two FIFO priority queues from the (congested) routers. Comparisons between these two schemes demonstrate that Monaco does not suffer from the problems mentioned above and achieves better performance than Vegas. We use extensive ns-2 simulations and Linux implementation experiments to show that the static and dynamic performance of Monaco matches the theoretic results. One key issue regarding the ACC model in general, i.e., the scalability of the bottleneck buffer requirement, and a solution using a virtual queuing algorithm, are discussed and evaluated.**

Keywords—**Congestion control, Accumulation, TCP.**

I. Introduction

Much research has been conducted to achieve stable, efficient and fair operation of packet-switching networks. TCP congestion control [16], a set of end-to-end mechanisms, has been widely acknowledged for its critical role in maintaining the stability of the Internet. Among them, TCP Reno [2] infers network congestion by detecting packet loss that is presumably caused only by congestion; alternatively, TCP Vegas [10] measures backlog, the number of buffered packets inside the routers along the path, to detect network congestion and demonstrates better performance than Reno. Unfortunately, Vegas has technical problems inherent to its backlog estimator that prevent it from functioning properly. There has been a substantial amount of work on this issue, such as [1] [25] [15] [8] [9] [24] [12] and references therein, which we review in Section II. But none of them provides a solution to estimate backlog unbiasedly in case of round trip propagation delay estimation error or reverse path congestion.

In this paper, we offer a solution to this problem and develop a systematic model to generalize Vegas' congestion avoidance mechanism. Formally, we define in a bit-by-bit fluid model the backlog (hereafter we call it *accumulation*) as a time-shifted, distributed sum of queue contributions of a flow at a set of FIFO routers on its path. The central idea is to control flows' rate by controlling their accumulations in an end-to-end and distributed manner. We study a set of closed-loop congestion control schemes that are all based upon the idea of keeping a target accumulation for each flow individually.

The key concepts for this accumulation-based congestion control (ACC) model are developed in Section III. An ACC model has two components: congestion estimation and congestion response. The former defines a congestion measure (i.e., accumulation) and provides an implementable estimation of the measure; while the latter defines an increase/decrease policy for the source throttle. A class of control algorithms, including the additive-increase/additive-decrease (AIAD) policy [11], Mo and Walrand's proposal [26] and a proportional control, can be used. Based on previous research [26] [18], in the appendix we show that the equilibria of all these algorithms achieve the same proportional fairness.

To instantiate the ACC model, choices can be made in each of the ACC components to put together the entire scheme. We describe two example schemes in Section IV. We show that Vegas congestion avoidance attempts to estimate accumulation and thus fits into the ACC family. But it often fails to provide an unbiased accumulation estimate. We therefore design a new scheme called Monaco that emulates the ACC fluid model in a better way. Particularly, Monaco solves the Vegas problems by employing an *out-of-band, receiver-based* accumulation estimator. We pro-

[1]This work was supported by the NSF contracts ANI-9806660 and ANI-9819112, by the DARPA contract F30602-00-2-0537, and a grant from Intel Corp. Part of the paper was presented at the IEEE International Conference on Communications in Anchorage, Alaska, USA on May 11-15, 2003.
Yong Xia and Shivkumar Kalyanaraman are with the ECSE department, Rensselaer Polytechnic Institute, Troy, NY 12180.
David Harrison is with the EECS department, University of California, Berkeley, CA 94720.
Kishore Ramachandran is with the WINLAB, Rutgers University, Piscataway, NJ 08855.
Arvind Venkatesan is with Bloomberg L.P., New York, NY 10022.

vide resolution to a number of concerns regarding accumulation estimation in Section IV-C. Section V demonstrates the steady state and dynamic performance of Monaco using extensive ns-2 [27] simulations as well as Linux implementation experiments. Section III-C discusses a key concern regarding the ACC model in general, i.e., the scalability of bottleneck buffer requirements resulting from accumulating packets in the congested router buffers for every flow. Section IV-D presents a solution to this issue based on the virtual queuing algorithm in [21]. We conclude this paper in Section VI.

II. Related Research

The most closely related work starts from the TCP Vegas protocol, followed by a series of nonlinear optimization based models for network congestion control.

TCP Vegas [10] includes three new techniques: a modified slow start, a more reactive retransmission mechanism resulting in less timeouts, and a new congestion avoidance that maintains a "right" amount of extra packets inside the network. Its authors claim that Vegas achieves higher throughput and less packet losses than Reno using simulations and Internet measurements, confirmed experimentally by Ahn et al. [1] and analytically by Mo et al. [25], who also point out Vegas' drawbacks of estimating round trip propagation delay (RTPD) incorrectly in the presence of rerouting and possible persistent congestion. Instead of using the minimum of all round trip time (RTT) samples as an estimation of RTPD, they suggest to use the minimum of only the most recent k RTT samples. As we discussed in Section IV-A, this estimation is still inflated because there is *always* a steady state standing queue on the path.

Bonald compares Reno and Vegas by means of a fluid approximation [8]. He finds that Vegas is more stable than Reno, resulting in a more efficient utilization of network resources, and shares fairly the available bandwidth between users with heterogeneous RTTs. But its fairness critically depends on accurate estimation of RTPD, confirmed by the analysis of Boutremans et al. [9].

A different line of research of network congestion control theoretic models is pioneered by Kelly's optimization framework [18], followed by Low et al. [23] and Srikant et al. [20], where they model congestion control as a nonlinear optimization problem under which all users try to maximize their own interest, subject to a set of capacity constraints. Following Gibbens and Kelly's work [13], Kunniyur and Srikant develop an Adaptive Virtual Queue (AVQ) algorithm [21], which we leverage in this paper to keep a low steady state queue in the congested routers (see Section IV-D). Low, Peterson and Wang present an

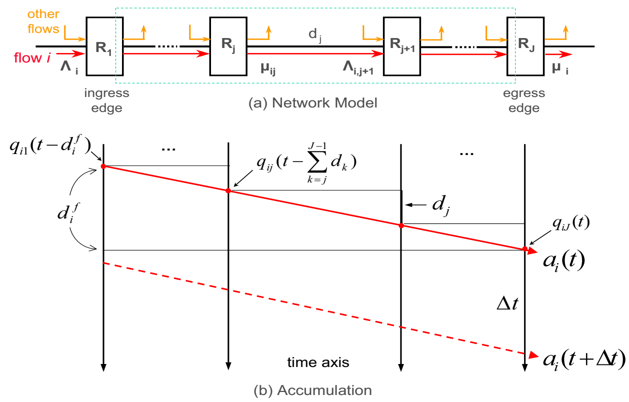


Fig. 1. Network Fluid Model of Accumulation

optimization model for Vegas [24]. Then Jin, Wei and Low extend Vegas and design a FAST protocol for high bandwidth-delay-product networks [17]. Low et al. improve Vegas performance using a Randomly Exponential Marking (REM) buffer management algorithm [3]. Similar to Vegas+REM, we use Monaco+AVQ in this paper as an alternative solution.

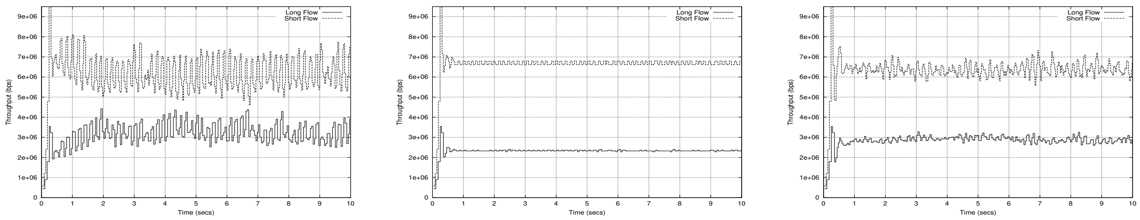
Mo and Walrand propose a fair end-to-end window-based scheme that includes a proportionally fair control algorithm [26]. However, this algorithm raises technical challenges in its practical implementation. Our Monaco accumulation estimator can be viewed as such an implementation that requires two-FIFO-priority-queue support from the congested routers.

III. ACC Fluid Model

In this section we describe the ACC model. We define accumulation under a bit-by-bit fluid model and use accumulation to measure and control network congestion. In the appendix we briefly prove that keeping a target accumulation inside the routers for each flow is equivalent to a nonlinear optimization that allocates network capacity proportionally fairly. We show that a set of control algorithms exist for each flow to achieve its target accumulation.

A. Accumulation

Consider an ordered sequence of FIFO nodes $\{R_1, \ldots, R_J\}$ along the path of a unidirectional flow i in Figure 1(a). The flow comes into the network at the ingress node R_1 and, after passing some intermediate nodes $R_2, \ldots, R_{J-1}$, goes out from the egress node R_J. At time t in any node R_j $(1 \leq j \leq J)$, flow i's input rate is $\lambda_{ij}(t)$, output rate is $\mu_{ij}(t)$. The propagation delay from node R_j to node R_{j+1}



(a) ACC-AIAD (b) ACC-MW (c) ACC-P

Fig. 2. Different ACC algorithms achieve similar steady state bandwidth allocation, but the dynamic behavior differs significantly.

is a constant value d_j.[2]

Define flow i's accumulation as a time-shifted, distributed sum of the queued bits in all the nodes along its path from the ingress node R_1 to the egress node R_J, i.e.,

$$a_i(t) = \sum_{j=1}^{J} q_{ij}(t - \sum_{k=j}^{J-1} d_k) \tag{1}$$

where $q_{ij}(t)$ is flow i's queued bits in router j at time t, as illustrated as the solid slanted line in Figure 1(b). Note the equation includes only those bits backlogged inside the node buffers, not those stored on transmission links. (This definition provides a reference to implement an unbiased accumulation estimator in Section IV-B.1.) We aim to control flow rates by controlling their accumulations, i.e., *keeping a steady state accumulation inside the network for each individual flow*.

B. Control Algorithms

In the ACC model we use accumulation to measure network congestion as well as to probe available bandwidth. If accumulation is low, we increase the congestion window; otherwise, we decrease it to drain accumulation. More accurately, we try to maintain a constant target accumulation a_i^* for each flow i and $a_i^* > 0$. To achieve this goal we can choose from a *set* of control algorithms:[3]

i) ACC-AIAD additively increases and additively decreases the congestion window value:

$$\dot{w}_i(t) = -\frac{\kappa}{rtt_i} \cdot sgn(a_i(t) - a_i^*) \tag{3}$$

[2]In practice R_1/R_J can be mapped as a source/destination pair to form an end-to-end control loop or ingress/egress edge routers to form an edge-to-edge control loop. Here we focus on the ACC model itself. The reader is refer to [14] for discussion on architectural issues.

[3]All these algorithms fit into the following general form:

$$\dot{w}_i(t) = -\eta \cdot g(t) \cdot f(a_i(t) - a_i^*) \tag{2}$$

where $\eta > 0$, $g(t) > 0$, $f(\cdot)$ is a function in the first and third quadrants. It is nondecreasing and has a single root 0 (i.e., only $f(0) = 0$).

where $w_i(t), rtt_i, a_i(t)$ and a_i^* are respectively the congestion window size, round trip time, instantaneous accumulation and target accumulation value of flow i, $\kappa > 0$ and

$$sgn(y) = \begin{cases} +1 & if \quad y > 0 \\ 0 & if \quad y = 0 \\ -1 & if \quad y < 0. \end{cases} \tag{4}$$

ii) ACC-MW is proposed by Mo and Walrand [26]:

$$\dot{w}_i(t) = -\frac{\kappa \cdot rtt_{pi}}{rtt_i} \cdot \frac{a_i(t) - a_i^*}{w_i(t)} \tag{5}$$

where rtt_{pi} is the round trip propagation delay of flow i.

iii) ACC-P is a proportional control policy that we use in this paper:

$$\dot{w}_i(t) = -\frac{\kappa}{rtt_i} \cdot (a_i(t) - a_i^*). \tag{6}$$

Note all the above algorithms have the same single zero point $a_i(t) = a_i^*$. We present a set of algorithms here because they share a common steady state property of proportionally fair bandwidth allocation. We briefly state this below and present more details in the appendix.

C. Properties

For any congestion control, major theoretic concerns are stability, as well as fairness and steady state queue bound. Stability guarantees that the algorithm approaches its equilibrium. Fairness, either max-min [7] or proportional [18], determines the steady state bandwidth allocation among the competing flows. A steady state queue bound provides an upper limit on the router buffer requirement in equilibrium, which is important for real network deployment.

The stability of the general algorithm (2) is still an open question. So we turn to extensive simulations in Section V to evaluate the stability of ACC-P.

Given that the equilibrium is achieved, we can prove that the equilibrium bandwidth allocation of ACC is weighted proportionally fair (See the appendix). Figure 2 shows simulation results of a parking-lot topology of two 9Mbps bottlenecks with one long flow and two short flows (using the scheme developed in Section IV-B). It verifies that all the three algorithms do achieve similar steady state bandwidth allocations.[4]

Interestingly, different ACC control policies can have the same fairness property. Thus to achieve a particular steady state performance, we have the freedom to choose from a set of control policies that have different dynamic characteristics. In this sense, the ACC model manifests congestion control in two steps: setting a target steady state allocation (fairness) and then designing a control algorithm (stability and dynamics) to achieve that allocation.

Even though we keep a finite accumulation inside the network for every flow, the steady state queue at a bottleneck scales up to the number of flows sharing that bottleneck. In practice, we need to provide enough buffer in the congested routers to avoid packet loss and make the congestion control protocol robust to such loss, if unavoidable (see Section IV-B). Another way to alleviate this problem is to control aggregate flow in a network edge-to-edge manner, instead of end-to-end. (However, we focus on the model itself and don't elaborate the architecture issues in this paper.) A possibly better solution to keep steady state queue length bounded is to use an active queue management (AQM) mechanism such as AVQ [21]. We implement this option and discuss more details in Section IV-D.

IV. ACC Schemes

Now we instantiate the ACC model into two schemes for packet-switching networks. Firstly we show that TCP Vegas tries to estimate accumulation and fits into the ACC model, but Vegas often fails to provide an unbiased accumulation estimate. Then we design a new scheme called Monaco that solves the estimation problems of Vegas. Monaco also improves the congestion response by utilizing the value of the estimated accumulation, unlike Vegas' AIAD policy that is slow in reacting to a sudden change in user demands or network capacity. By comparing Monaco and Vegas via analysis and simulation we reach two observations: It is effective 1) to employ a *receiver-based* mechanism and, 2) to measure *forward path queuing delay*, instead of round trip queuing delay as in Vegas, for an unbiased accumulation estimate. The scheme design is guided by the following goals:

[4]More careful investigation of Figure 2 reveals that the equilibria of the three algorithms are not exactly the same. We believe that it is due to the burstiness in the discrete time simulation system which is not captured by the continuous time fluid model.

Goal 1: **Stability:** The scheme should steer toward an equilibrium in a reasonably dynamic environment with changing demands or capacity;

Goal 2: **Proportional Fairness:** Given enough buffers, the scheme must achieve proportional fairness and operate without packet loss at the steady state;

Goal 3: **High Utilization:** When a path is presented with sufficient demand, the scheme should achieve around full utilization of the path's resources;

Goal 4: **Avoidance of Persistent Loss:** If the queue should grow to the point of loss due to underprovisioned buffers, the scheme must back off to avoid persistent loss.

A. Vegas

Vegas includes several modifications over Reno. However, we focus only on its congestion avoidance mechanism, which fits well as an example ACC scheme.

Brakmo and Peterson [10] use the term "backlog" to describe what we explicitly define as accumulation. For each flow, the Vegas estimator takes as input an estimate of its round trip propagation delay, hereafter called rtt_p (or $basertt$ in [10] [25]). Vegas then estimates the accumulation as

$$\begin{aligned} \hat{a}_V &= (expected\, rate - actual\, rate) \times rtt_p \\ &= (\frac{cwnd}{rtt_p} - \frac{cwnd}{rtt}) \times rtt_p \end{aligned} \quad (7)$$

which could be simplified as

$$\hat{a}_V = \frac{cwnd}{rtt} \times rtt_q \quad (8)$$

where $cwnd/rtt$ is the average sending rate during that RTT and $rtt_q = rtt - rtt_p$ is the round trip queuing delay. If rtt_p is accurately available and there is no reverse path queuing delay, then according to Little's Law, $\hat{a}_V$ provides an unbiased accumulation estimate.

Vegas estimates rtt_p as the minimum RTT measured so far. If the bottleneck queues drain often, it is likely that each control loop will eventually obtain a sample that reflects the true propagation delay. The Vegas estimator is used to adjust its congestion window size, $cwnd$, so that $\hat{a}_V$ approaches a target range of ε_1 to ε_2 packets. More accurately stated, the sender adjusts $cwnd$ using a variant version of the algorithm (3):

$$cwnd(n+1) = \begin{cases} cwnd(n) + 1 & if \quad \hat{a}_V < \varepsilon_1 \\ cwnd(n) - 1 & if \quad \hat{a}_V > \varepsilon_2 \end{cases} \quad (9)$$

where ε_1 and ε_2 are set to 1 and 3 packets, respectively. Vegas has several well-known problems:

• Rtt_p *Estimation Errors*: Suppose re-routing of a flow increases its propagation delay. Vegas misinterprets such an increase as less congestion and sends faster. Hence, this policy can lead to unbounded queue which introduces persistent loss and congestion [24], violating Goals 1 and 4. Mo et al. [25] suggest limiting the history on the rtt_p estimate by using the minimum of the last k, instead of all, RTT samples. We refer to this variant as the "Vegas-k" scheme. Still, it cannot guarantee queue drain at intermediate bottlenecks within k RTTs, shown in Section IV-C.

• Rtt_p *with Standing Queues*: When a flow arrives at a bottleneck with a standing queue, it obtains an exaggerated rtt_p estimate. The flow then adjusts its window size to incur an extra backlog between ε_1 and ε_2 packets in addition to the standing queue. This leads to a bandwidth allocation away from the target proportional fairness, violating Goal 2.

• *Reverse Path Congestion*: The Vegas estimator is affected by congestion in the reverse path. Reverse path congestion inflates the Vegas estimator leading to sharply reduced utilization, not achieving Goal 3.

B. Monaco

Monaco emulates the accumulation defined by Equation (1) and implements a receiver-based, out-of-band measurement. It is immune to issues such as rtt_p sensitivities and reverse path congestion and robust to control and data packet losses. We describe the Monaco accumulation estimator and then its congestion response policy.

B.1 Congestion Estimation Protocol

Let's look at the definition of accumulation in Equation (1). It is the sum of the queued bits of a flow at a sequence of FIFO routers, including both ingress and egress nodes as well as the intermediate routers. We aim to eliminate the computation at intermediate routers. Actually it is impossible for all the nodes R_j $(1 \leq j \leq J)$ to compute synchronously their queues $q_{ij}(t - \sum_{k=j}^{J-1} d_k)$ at different times since no common clock is maintained.

To estimate accumulation without explicit computation at the intermediate routers, Monaco generates a pair of back-to-back control packets once per RTT at the ingress node as shown in Figure 3. One control packet is sent out-of-band (OB) and the other in-band (IB). The OB control packet skips queues in the intermediate routers by passing through a separate dedicated high priority queue.[5] Assuming the OB queues to be minimal as only other OB control packets share them, such packets experience only the forward propagation delay d_i^f. The IB control packet goes along with regular data packets and reaches the egress node after experiencing the queuing delay in the network. The time interval between the OB and IB control packets measured at the egress node is a sample of the current forward trip queuing time (ftt_q). Considering a network with enough buffers where there is no packet loss, if the flow rates at all the routers do not change dramatically, then by Little's Law, the number of data packets arriving at the egress node after the OB control packet, but before the IB control packet is an unbiased accumulation estimate. In Figure 3, the dashed lines cut by the forward direction OB control packet are those data packets, with each cut happening in the router R_j at time $t - \sum_{k=j}^{J-1} d_k, \forall j \in \{1, ..., J\}$. Also observe in the figure that we can measure rtt at both ingress and egress nodes and rtt_p at the egress node.

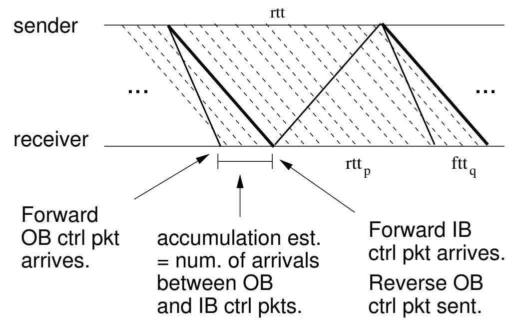


Fig. 3. Monaco Accumulation Estimator

Besides, we need to consider the effect of traffic burstiness. When we have a congestion window size $cwnd$, we also compute a rate based on RTT estimation: $rate = cwnd/rtt$. At the ingress node we use this rate value to smooth incoming traffic and thus alleviate the effect of burstiness. At the egress node the accumulation is computed as the product of ftt_q and an exponentially weighted moving average of the egress rate.

In practice, both data and control packets maybe lost because of inadequate router buffer size or too many competing flows. To enhance the robustness of the Monaco estimator when data packets are lost, the IB control packet, identified by a control packet sequence number, carries a byte count of the number of data bytes sent during that period. If the egress node receives fewer bytes than were transmitted, then packet loss is detected. The forward OB control packet carries the same control packet sequence

[5] An alternative implementation is to use IP Type of Service (TOS), i.e., assigning a low delay TOS to the high priority control packet if TOS is supported in all the (congested) routers.

number as the associated IB control packet. Monaco sends congestion feedback on the reverse OB control packet, in which there is one additional piece of information: a flag denoting whether the congestion window $cwnd$ should increase, decrease, or decrease-due-to-loss. The subsequent pair of forward control packets is generated after the arrival of the reverse OB control packet at the ingress node.

If either IB or OB control packet is lost, then the ingress node times out and sends a new pair of control packets with a larger sequence number. The timer for control packet retransmission is similar to that of TCP. These routine reliability enhancements are similar to those in the Congestion Manager [4].

B.2 Congestion Response Protocol

As already noted, we use accumulation to measure network congestion and to probe available bandwidth. We keep target accumulation for each flow by increasing/decreasing its congestion window when the accumulation is lower/higher than the target value.

Since a pure window-based control policy introduces undesirable burstiness we use *rate-paced window control* to smooth incoming traffic by employing at the ingress node a leaky bucket shaper with a rate value of $cwnd/rtt$ and a burst parameter of one packet.

We provide below the Monaco's proportional control policy which is the discrete version of Equation (6):

$$cwnd(n+1) = cwnd(n) - \kappa \cdot (\hat{a}_M - a^*) \qquad (10)$$

where $\hat{a}_M$ is the Monaco accumulation estimation, a^*, set to 3 packets, is a target accumulation in the path akin to ε_1 and ε_2 used by Vegas, κ is set to 0.5, and $cwnd(n)$ is the congestion window value at a control period n.

Monaco improves Vegas' control policy by utilizing the value of estimated accumulation fedback by the reverse OB control packet, instead of taking it as binary information (i.e., "how congested", instead of "congested or not"). If the congestion feedback is decrease-due-to-loss, Monaco halves the congestion window as in TCP Reno.

C. Comparisons between Vegas and Monaco

Vegas and Monaco both aim to accurately estimate accumulation, assuming different support from the network routers. If rtt_p can be obtained precisely and there is no reverse path congestion, then, by Little's law, each of them gives an unbiased accumulation estimate. But in practice Vegas has severe problems in achieving this objective; Monaco solves known estimation problems.

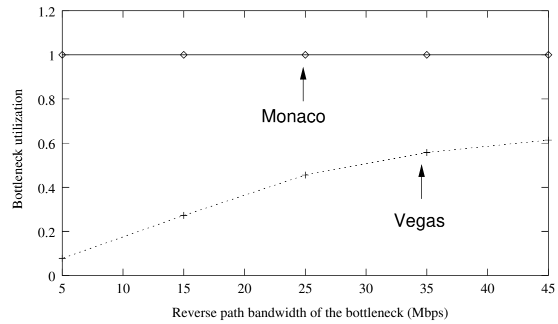


Fig. 4. Comparison between Vegas and Monaco under Reverse Path Congestion

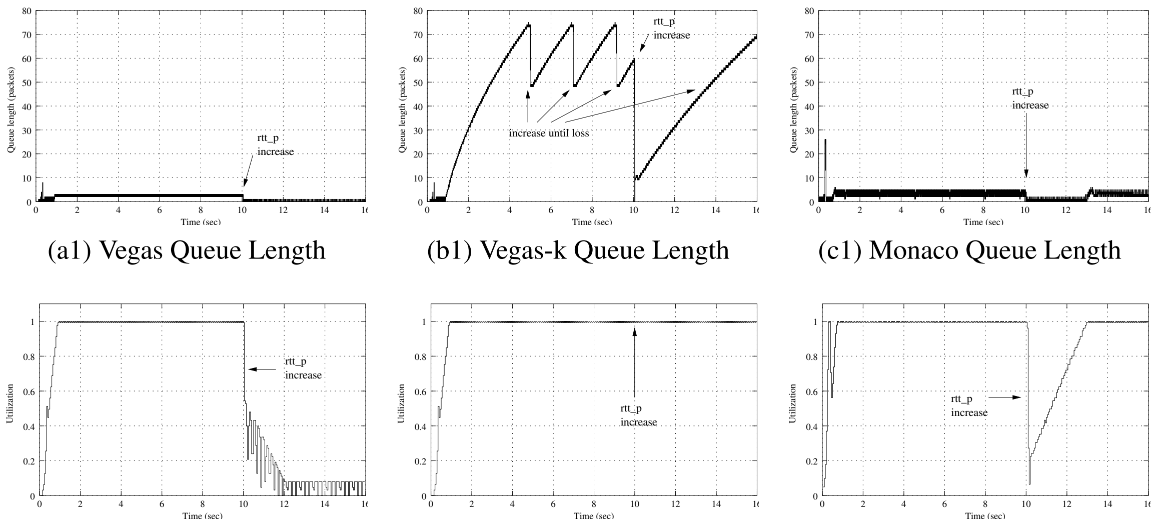
Vegas estimator operates at the *sender* side. According to Equation (8) it actually calculates:

$$\hat{a}_V = \frac{cwnd}{rtt} \times (rtt - rtt_p) \qquad (11)$$

$$= \frac{cwnd}{rtt} \times (t_q^f + t_q^b) \qquad (12)$$

where t_q^f and t_q^b are the forward and reverse path queuing delays, respectively. The above equations imply that Vegas may suffer from two problems: 1) By Equation (12), if there exists reverse path queuing delay (because of reverse direction flows), i.e., $t_q^b > 0$, then Vegas overestimates accumulation. This leads to underutilization and is hard to handle because the forward direction flows have no control over those in the reverse direction. To show this effect we use a simple dumb-bell topology with a bottleneck of 45Mbps forward direction bandwidth shared by seven forward direction flows and seven reverse flows. We change the bottleneck's reverse direction bandwidth from 5Mbps to 45Mbps. As shown in Figure 4, Vegas utilization is only 10% $\sim$ 60%. 2) By Equation (11), if rtt_p is underestimated then Vegas overestimates accumulation and conversely if rtt_p is overestimated. The prior leads to the affected flows reducing window size leading to unfairness or underutilization. Results for a single bottleneck of 10Mbps bandwidth and 12ms propagation delay are shown in Figures 5(a) and 5(b), where the rtt_p estimation error is introduced by a sudden propagation delay change to 52ms at time 10s. Vegas retains its prior now inflated rtt_p estimate and thus underutilizes the bottleneck. Vegas-k recovers from the increase in propagation delay, but because the queue does not periodically drain, the minimum of the last k RTTs drifts upward overestimating rtt_p until buffer overflows.

Due to the above problems, Vegas falls short of qualifying as an effective ACC scheme, because we expect to achieve congestion control by maintaining constant accumulation for each flow at the steady state! In such a case,



(a1) Vegas Queue Length (b1) Vegas-k Queue Length (c1) Monaco Queue Length

(a2) Vegas Utilization (b2) Vegas-k Utilization (c2) Monaco Utilization

Fig. 5. Comparison between Vegas, Vegas-k and Monaco under rtt_p (or $basertt$) Estimation Error

the sum of accumulations would lead to a non-zero steady state queue which is not likely to drain, and hence dynamic rtt_p estimation would not possibly be unbiased with only in-band measurements. In summary, the rtt_p sensitivity issues of Vegas point to a *fundamental* problem of the in-band techniques for accumulation estimation.

Monaco solves both problems. Monaco estimator operates at the *receiver* side and thus excludes the influence of reverse path congestion. By measuring the time interval between the OB and IB control packets, Monaco does not need to explicitly estimate the forward path propagation delay. (Actually the forward path OB control packet implicitly provides this value.) More specifically, since Monaco implements a rate-paced window control algorithm to smooth out incoming traffic, the time difference between the OB and IB control packet arrivals gives a sample of the current forward path queuing delay ftt_q. By Little's law, the number of data packets arriving during this time period is the backlogged packets along the path. Using the OB control packet also makes Monaco adaptive to re-routing since it is sent every RTT. As shown in Figures 4 and 5(c), after a brief transient period of three seconds, Monaco operates again at around 100% utilization with no packet loss. So it's immune to rtt_p estimation inaccuracy and reverse path congestion.

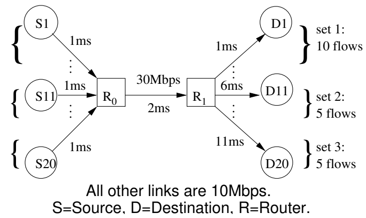
The above comparisons between Vegas, Vegas-k and Monaco suggest two observations on how to estimate accumulation unbiasedly: 1) The key is to measure *forward path* queuing delay (via the OB and IB control packets in Monaco), instead of *round trip* queuing delay (as in Vegas); and consequently, 2) it's better to measure accumulation at the *receiver side*, otherwise it's difficult to eliminate the effect of reverse path queuing delay, which is hardly under the forward direction congestion control.

D. Adaptive Virtual Delay Queuing

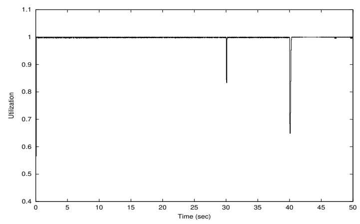
As we discussed in Section III-C there is a buffer scalability problem for *all* the ACC schemes, including Vegas, Monaco and FAST, since all of them keep non-zero steady state accumulation inside the network for all the flows.

According to the analysis in the appendix, the key to all the ACC schemes is to provide the queuing delay, or the Lagrange multiplier from the optimization perspective, that is a measure of network congestion. In a non-AQM droptail FIFO router, the Lagrange multiplier $t_{ql} = q_l/c_l$ is provided by a *physical* FIFO queuing process where c_l is fixed and we have no freedom to control the physical queue q_l. But similar to AVQ, we can provide the same value of the Lagrange multiplier t_{ql} by running an AQM algorithm in the bottleneck such that $t_{ql} = q'_l/c'_l$ if we adapt the virtual capacity c'_l appropriately (Also see discussions in [12]). At the same time the physical queue q_l can be bounded.

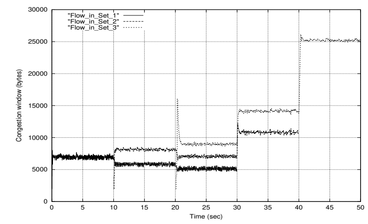
So we leverage AVQ to emulate an adaptively changing link capacity and compute a virtual queuing delay, which is defined as the ratio of virtual queue length divided by virtual capacity and add it into the forward IB control packet. We call this mechanism Adaptive Virtual Delay (AVD) algorithm. A nice property of AVD is that it is *incrementally deployable* since a mixed set of droptail and AVD routers can work together (see Section V-B). In such an environment the Monaco accumulation estimate changes



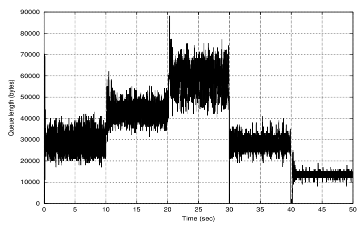
(a) Topology



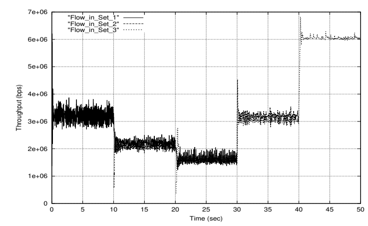
(b) Bottleneck Utilization



(c) Flow Congestion Window

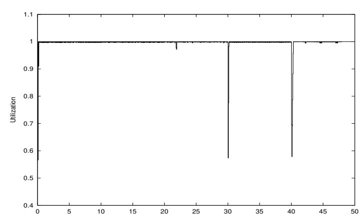


(d) Bottleneck Queue Length

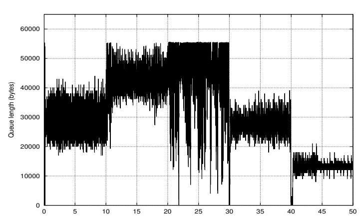


(e) Flow Throughput

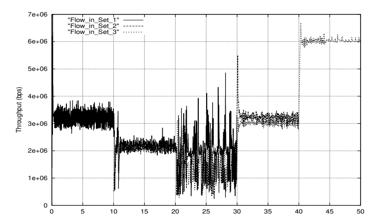
Fig. 6. Monaco with Enough Buffer (90 packets) in a Droptail Bottleneck



(a) Bottleneck Utilization

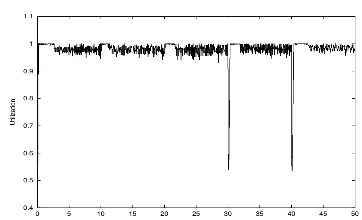


(b) Bottleneck Queue Length

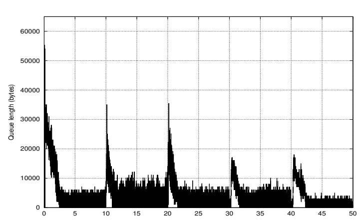


(c) Flow Throughput

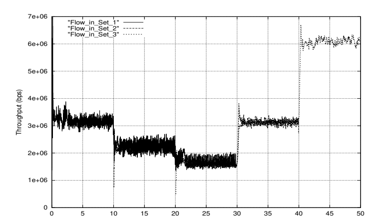
Fig. 7. Monaco with Underprovisioned Buffer (55 packets) in a Droptail Bottleneck



(a) Bottleneck Utilization



(b) Bottleneck Queue Length



(c) Flow Throughput

Fig. 8. Monaco with the Same Buffer as the Above Case (55 packets) in an AVD Bottleneck

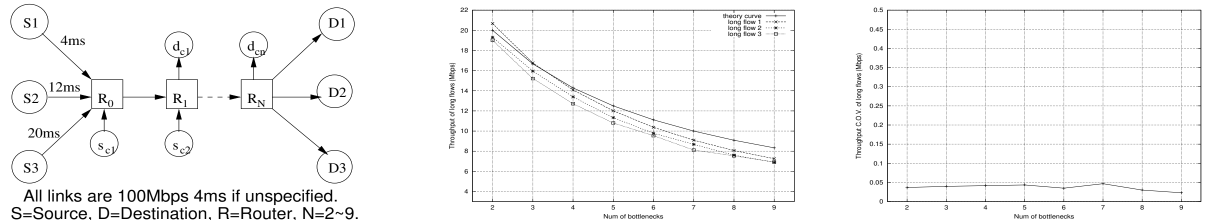
to $\hat{a}_M = \hat{a}_{DT} + x \cdot \hat{t}_{AVD}$, where $\hat{a}_{DT}$ is the accumulation in the droptail bottlenecks measured between two control packets as in Figure 3, x is the egress flow rate and $\hat{t}_{AVD}$ is the sum of all the virtual delays at those AVD bottlenecks.

V. Simulations and Experiments

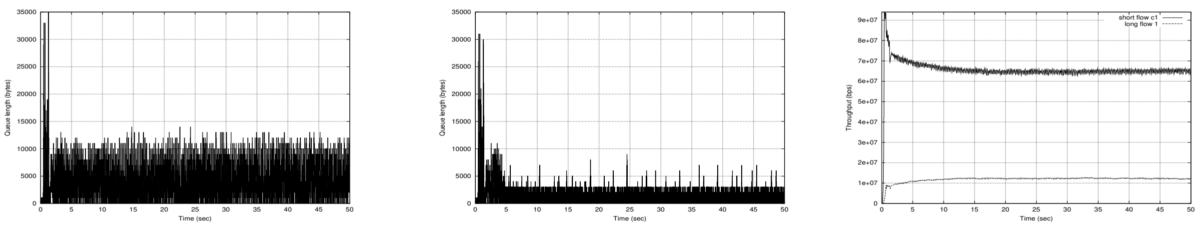
In the last section we have shown that Monaco outperforms Vegas. So we focus on evaluating the Monaco scheme using simulations and implementation experiments in this section. Our ns-2 simulations illustrate:

A) Dynamic behaviors such as convergence of throughput, instantaneous link utilization and queue length in Section V-A. We use a single bottleneck topology with heterogeneous RTTs for tens of flows periodically entering and leaving;

B) Steady state performance such as throughput fairness in Section V-B. We use a linear topology of multiple congested links shared by a set of flows passing different number of droptail and AVD bottlenecks.



(a) Topology (b) Long Flow Average Throughput (c) Throughput Spread



(d) Droptail Bottleneck Queue Length (e) AVD Bottleneck Queue Length (f) Throughput Convergence

Fig. 9. Monaco without Background Traffic under Multiple Mixed Droptail and AVD Bottlenecks

TABLE I
PARAMETER SETTINGS

Parameter	Value
κ	0.5
target accumulation, a^*	3000 bytes
data packet size	1000 bytes

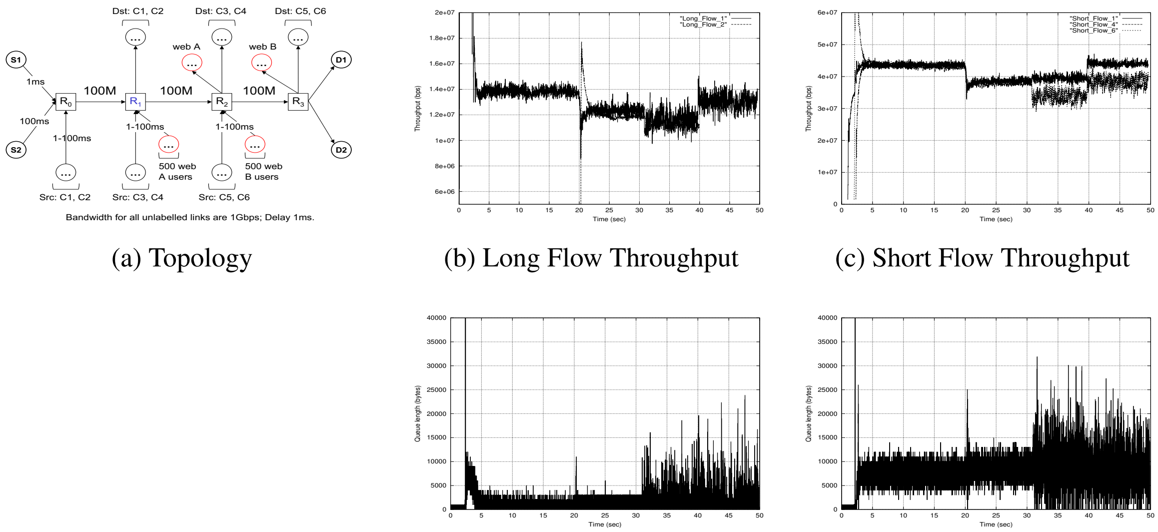
We also implement Monaco in Linux kernel v2.2.18 based on the Click router [19]. In Section V-C we use Linux implementation experiments to validate the ns-2 simulation results. In all the simulations and experiments we use the parameter settings shown in Table I. In brief, in combination with Section IV-C, this section shows that Monaco satisfies all the goals outlined in the beginning of Section IV.

A. A Single Bottleneck with Dynamic Demands

Firstly we consider a single 30Mbps bottleneck with 2ms propagation delay shared by three sets of flows using Monaco, as shown in Figure 6(a). Set 1 has ten flows starting at 0s and stopping at 30s; Set 2 has five flows starting at 10s and stopping at 40s; Set 3 has five flows starting at 20s and stopping at 50s. Each source-destination pair is connected to the bottleneck via a 10Mbps 1ms link. The one-way propagation delays for the three sets of flows are 4ms, 9ms and 14ms, respectively. We simulate for 50 seconds. We perform three simulations, the first one with enough buffer provided for a droptail bottleneck, the second one with underprovisioned buffer also for the droptail bottleneck, and the third one with an AVD bottleneck.

In the first simulation, the bottleneck router has enough buffer of 90 packets, as shown in Figure 6(d), where there is no packet loss. We randomly pick one flow from each set and draw its individual throughput in Figure 6(e). We observe that from 0s to 30s, the throughput is about 3Mbps, since only ten flows are active. When the five flows from set 2 jump in at 10s, the throughput drops to 2Mbps, as we have fifteen active flows. Similarly, when the final five flows from set 3 enter at 20s, the throughput changes to 1.5Mbps. Then at 30s, set 1 stops, the throughput increases to 3Mbps. At 40s, set 2 leaves, only the five flows of set 3 are in the system with throughput of about 6Mbps. The congestion window dynamics is similar, as shown in Figure 6(c). Bottleneck queue length is depicted in Figure 6(d) where the incoming flows build up a steady queue and the flows leave with queue decrease, on average 3 packets for each flow. This matches the target accumulation specified as a control parameter in Table I. During the simulation the bottleneck utilization always stays around 100%, except two soon-recovered drops during abrupt demand changes at 30s and 40s as seen in Figure 6(b). From this simulation, we validate that Monaco demonstrates a stable behavior under a dynamic and heterogeneous environment and keeps steady queues inside the bottleneck.

In the second simulation, the droptail bottleneck router buffer is underprovisioned, as illustrated in Figure 7(b), we can see that the queue length grows to the limit of the whole buffer size of 55 packets, and there is a corresponding packet loss leading to halving of the congestion window during 20s $\sim$ 30s. Consequently, the throughput oscillates more as seen in Figure 7(c), but the bottleneck



(a) Topology (b) Long Flow Throughput (c) Short Flow Throughput

(d) R_1 AVD Queue Length (e) R_2 Droptail Queue Length

Fig. 10. Monaco with a Large Amount of Background Web Traffic under Multiple Mixed Droptail and AVD Bottlenecks

is still fully utilized. From this simulation, we see that without enough buffer, Monaco shows a degraded behavior under dynamically changing demands.

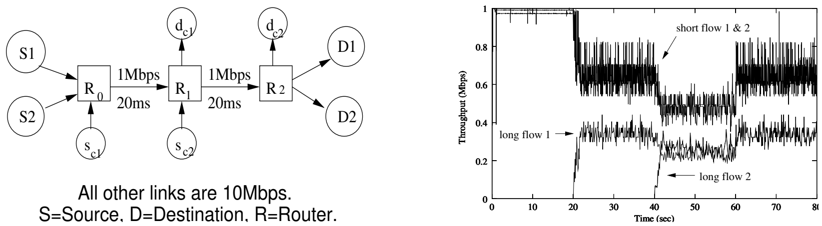
In the third simulation, the AVD bottleneck buffer is the same as the second one. As illustrated in Figure 8(b), for most of time the bottleneck queue length is below 10 packets. The throughput converges without oscillation as shown in Figure 8(c), comparable to result in the first simulation. The bottleneck utilization is around 98%, which is the target utilization value we configure in the AVD algorithm. This simulation shows that the AVD mechanism is effective in controlling the queue size and thus makes Monaco more stable comparing to the droptail bottleneck without enough buffer provisioned.

B. Multiple Bottlenecks

Firstly we show the steady state performance of Monaco when a flow traverses more than one bottleneck. We use a linear topology with multiple congested links depicted in Figure 9(a). We perform a series of simulations by changing the number of the bottlenecks N from 2 to 9. To show the compatibility of AVD in a droptail environment, we randomly set some bottlenecks droptail and others AVD. There are three "long" flows passing all the bottlenecks and a set of "short" flows each using only one bottleneck. Every bottleneck link has 100Mbps capacity and 4ms delay. The long flows have very different RTTs. We simulate for 50 seconds under only one condition with enough buffer provided for all the droptail routers.

As illustrated in Figure 9(b), the steady state throughput curves of all long flows are located near the theoretic curve for proportional fairness, $100/(3+N)$Mbps. Each individual long flow gets roughly its fair share, for all cases of $N = 2, 3, ..., 9$ bottlenecks. The difference of throughput between the 3 long flows is measured by the Coefficient of Variance (C.O.V.) of their throughput, depicted in Figure 9(c), which is between 2% and 5% for all the cases. For a particular simulation of five bottlenecks, we pick up two of them, one droptail and one AVD, and draw their queue length in Figures 9(d) and 9(e), respectively. Obviously the AVD bottleneck keeps a lower queue than the droptail. We show the throughput convergence of two kinds of flows in Figure 9(f), where after about 10s of transient period, the long flow's throughput converges to 12Mbps (around its theoretic fair share of 12.5Mbps), and the short flow's to some 65Mbps (around its theoretic fair share of 62.5Mbps). This simulation demonstrates that, with enough buffer provisioned, Monaco achieves a proportionally fair bandwidth allocation in a multiple bottleneck case, validating our theoretic results in Section III.

Now we go further for a more realistic condition by adding web traffic into the former multiple bottleneck network. To simulate web traffic, we use Barford and Crovella's HTTP model introduced in [5]. A three-bottleneck topology is shown in Figure 10(a) where R_1 is an AVD router and others are droptail. All the bottleneck bandwidth is 100Mbps, whereas the access bandwidth is 1Gbps. The propagation delay for each link is also shown in the figure. Note "1-100ms" means that there are a number of links with propagation delays evenly ranging from 1ms to 100ms. All the unlabeled links have 1ms delay. There are two long flows, three short-flow sets each of



(a) Topology (b) Flow Throughput

Fig. 11. Monaco Linux Kernel Implementation Results

them includes two flows, and two web-traffic sets which totally have one thousand web connections multiplexed on the fifty access links. Long flow 1 starts at 0s and stops at 50s, while long flow 2 starts at 20s and stops at 40s. All short flows start randomly during 0∼3s. Web traffic starts at 30s. We simulate for 50 seconds.

The throughput dynamics for the two long flows and three randomly chosen short flows is shown in Figures 10(b) and 10(c) which demonstrate that both long and short flows' throughput rates converge respectively onto their theoretic values, i.e., $\frac{1\ (or\ 3)\times 100Mbps}{6+number\_of\_active\_long\_flows}$ for the long (or short) flows. For example, the long flow 1 gets 14Mbps during 0∼20s, 12.4Mbps during 20s∼30s, drops a little when web traffic comes in and then goes up to about 13Mbps after the long flow 2 leaves at 40s; whereas the short flow 1 gets 43Mbps during 3∼20s, 38Mbps during 20∼30s, and then a little more than those because the web traffic at the bottlenecks $R_1 \rightarrow R_2$ and $R_2 \rightarrow R_3$ limits the throughput of the long flows. The queue length of the AVD bottleneck R_1 is shown in Figure 10(d), while the droptail bottleneck R_2's is depicted in Figure 10(e). Obviously AVD keeps a low and constant queue, while the droptail queue is sensitive to the number of flows. Even when the web traffic jumps in, which makes the AVD queue more oscillating, it is still much lower than the droptail queue. In summary, this simulation shows that Monaco works in an environment of multiple bottlenecks with dynamic demands and bursty background traffic.

C. Implementation Results

We did a set of experiments using our Monaco Linux implementation to validate the stability and fairness results from the ns-2 simulations in the last two subsections. Here we show one result for a two-bottleneck topology with dynamic demands. For more details, the reader is referred to [28]. We have 2 droptail bottlenecks each of 1Mbps bandwidth and 20ms delay as drawn in Figure 11(a). During the 80s experiment, we have 2 short flows always active, one long flow coming in at 20s and going out at 60s, and another long flow active from 40s to 80s. After a brief transient period, each flow stabilizes at its proportionally fair share, illustrated by Figure 11(b). For instance, the first long flows' throughput starts with 0.33Mbps (its proportionally fair share) at 20s and changes to some 0.25Mbps at 40s when the second long flow shows up. At the same time, the short flows get 0.5Mbps, dropping from their former throughput of about 0.65Mbps. After 60s, the second long flow gets about its fair share of 0.33Mbps.

Comparing with the simulation throughput results in Section V-A, the implementation results in Figure 11(b) oscillate more. This comes mainly from the limited timer granularity in the Linux kernel which makes traffic less regulated (more bursty) than in ns-2.

VI. Summary

In this paper we generalize TCP Vegas and develop a general model using accumulation, buffered packets of a flow inside the network routers, as a measure to detect and control network congestion. Applying Mo and Walrand's queuing analysis and Kelly's nonlinear optimization framework on the model (in the appendix), we show that ACC allocates network bandwidth proportionally fairly at the equilibrium – which is its steady state feature. A set of control algorithms can drive the network to the same equilibrium – this is related to its dynamic characteristics. A family of schemes, including Vegas, could be derived based on ACC. Using the model as a reference, we design a new Monaco scheme that, with two FIFO priority queues provided by the (congested) network routers, solves the well-known problems of Vegas. In particular, using analysis and simulations, we show that the receiver-based, out-of-band estimator is able to produce an unbiased accumulation measurement.

Using extensive ns-2 simulations, we evaluate the dynamic and steady state performance of Monaco under different topologies and conditions. The scheme demonstrates its effectiveness in keeping network stable, fair, and efficiently utilized, given enough buffers in the bottlenecks. With underprovisioned buffer, Monaco's performance degrades. This buffer scalability problem can be solved by running the AVQ algorithm inside the bottlenecks, which works compatibly with the non-AQM droptail routers. We implement Monaco in Linux kernel based on the Click router and validate most of the simulation results on an internal testbed and the Utah Emulab [22].

In summary, the main contributions of this paper are:

- a model of accumulation-based congestion control based on which a family of schemes could be derived;
- a Monaco scheme implemented as a packet-switching network protocol which estimates accumulation unbiasedly and utilizes this value in a non-binary manner to control congestion;
- a comparison between Vegas and Monaco showing that Monaco's receiver-based, out-of-band accumulation measurement solves Vegas' well-known estimation problems;
- an incrementally deployable virtual delay queuing algorithm based on AVQ as a solution to the problem of unbounded bottleneck buffer size requirement.

One may ask that if the two-queue support from all the bottlenecks, even its complexity is very low, is unrealistic. Firstly, this requirement is largely eliminated at an AVD bottleneck which provides virtual delay, instead of using physical queuing delay, as its congestion measure. Secondly, for a non-AQM droptail bottleneck, as already explored in related research and this paper, in-band measurement techniques suffer from inherently hard accumulation estimation problem. So there is a fundamental tradeoff between an ACC scheme's performance and its requirements.

By keeping different accumulation for different flows, it's possible to provide differentiated services, such as minimum bandwidth assurance and weighted rate differentiation. These issues are explored in our related work [14].

VII. Acknowledgment

The authors would like to thank Prof. Steven Low of Caltech for discussions on REM. Thanks to their colleagues at RPI: Josh Hort, Sthanu Ramakrishnan and Rahul Sachdev for their related work, Prof. Murat Arcak and Xinzhe Fan for helpful discussions, and Prof. Biplab Sikdar and Kerry Wood for proofreading a draft version of this paper. They are grateful to the anonymous reviewers and the editor for their insightful comments.

References

[1] J. Ahn, P. Danzig, Z. Liu and L. Yan. Evaluation of TCP Vegas: Emulation and Experiment. *Proc. SIGCOMM'95*, August 1995.

[2] M. Allman, V. Paxson and W. Stevens. TCP Congestion Control. *IETF RFC 2581*, April 1999.

[3] S. Athuraliya, V. Li, S. Low and Q. Yin. REM: Active Queue Management. *IEEE Network*, 15(3):48-53, May 2001.

[4] H. Balakrishnan, H. Rahul and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. *Proc. SIGCOMM'99*, September 1999.

[5] P. Barford and M. Crovella. A Performance Evaluation of Hyper Text Transfer Protocols. *Proc. SIGMETRICS'99*, March 1999.

[6] M. Bazaraa, H. Sherali and C. Shetty. Nonlinear Programming: Theory and Algorithms. 2nd Ed., John Wiley & Sons, 1993.

[7] D. Bertsekas and R. Gallager. Data Networks. 2nd Ed., Simon & Schuster, December 1991.

[8] T. Bonald. Comparison of TCP Reno and TCP Vegas via Fluid Approximation. *INRIA Tech Report*, November 1998.

[9] C. Boutremans and J. Le BoudecT. A Note on the Fairness of TCP Vegas. *Proc. of International Zurich Seminar on Broadband Communications*, February 2000.

[10] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465-1480, October 1995.

[11] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, 17(1):1-14, June 1989.

[12] D. Choe and S. Low. Stabilized Vegas. *Proc. INFOCOM'03*, April 2003.

[13] R. Gibbens and F. Kelly. Resource Pricing and the Evolution of Congestion Control. *Automatica*, 35:1969-1985, 1999.

[14] D. Harrison, Y. Xia, S. Kalyanaraman and A. Venkatesan. A Closed-loop Scheme for Expected Minimum Rate and Weighted Rate Services. *Accepted by Computer Networks Journal*, March 2004. Http://www.rpi.edu/~xiay/pub/acc_qos.ps.gz.

[15] U. Hengartner, J. Bolliger and T. Gross. TCP Vegas Revisited. *Proc. INFOCOM'00*, March 2000.

[16] V. Jacobson. Congestion Avoidance and Control. *Proc. SIGCOMM'88*, August 1988.

[17] C. Jin, D. Wei and S. Low. FAST TCP: Motivation, Architecture, Algorithms, Performance. *Proc. INFOCOM'04*, March 2004.

[18] F. Kelly, A. Maulloo and D. Tan. Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, Vol.49, pp. 237-252, 1998.

[19] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek. The Click Modular Router. *ACM Trans. on Computer Systems* 18(3):263-297, August 2000.

[20] S. Kunniyur and R. Srikant. End-To-End Congestion Control: Utility Functions, Random Losses and ECN Marks. *Proc. INFOCOM'00*, March 2000.

[21] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. *Proc. SIGCOMM'01*, August 2001.

[22] J. Lepreau et al. The Utah Emulab. Http://www.emulab.net/.

[23] S. Low and D. Lapsley. Optimization Flow Control, I: Basic Algorithm and Convergence. *IEEE/ACM Trans. on Networking*, 7(6):861-875, December 1999.

[24] S. Low, L. Peterson and L. Wang. Understanding TCP Vegas: A Duality Model. *Proc. SIGMETRICS'01*, June 2001.

[25] J. Mo, R. La, V. Anantharam and J. Walrand. Analysis and Com-

parison of TCP Reno and Vegas. *Proc. INFOCOM'99*, March 1999.
[26] J. Mo and J. Walrand. Fair End-to-End Window-based Congestion Control. *IEEE/ACM Trans. on Networking*, 8(5):556-567, October 2000.
[27] Network Simulator ns-2. Http://www.isi.edu/nsnam/ns/.
[28] A. Venkatesan. An Implementation of Accumulation-based Congestion Control Schemes. *RPI M.S. Thesis*, August, 2002.

APPENDIX

We apply Mo and Walrand's queuing analysis [26] and Kelly's nonlinear optimization framework [18] to demonstrate the equilibrium characteristics of the ACC model described in Section III. It turns out that, given an appropriate control algorithm, ACC steers the network to an equilibrium of proportional fairness.

Network congestion control can be formalized as a resource allocation problem. Consider a network of a set $L = \{1, \ldots, |L|\}$ of links, shared by a set $I = \{1, \ldots, |I|\}$ of flows. Each link $l \in L$ has capacity c_l. Flow $i \in I$ passes a route L_i consisting of a subset of links, i.e., $L_i = \{l \in L \mid i \text{ traverses } l\}$. A link l is shared by a subset I_l of flows where $I_l = \{i \in I \mid i \text{ traverses } l\}$. Obviously $l \in L_i$ if and only if $i \in I_l$.

Firstly consider the queuing perspective [26]. After the system approaches a steady state (so we can neglect the time variable t in all the equations in Section III), at any link l the queue length q_l $(= \sum_{i \in I_l} q_{il})$, or equivalently the queuing delay t_{ql} $(= q_l/c_l)$, could be non-zero *only* if the capacity c_l is fully utilized by the sharing flows of the aggregate rate $\sum_{i \in I_l} x_i$, where x_i is the sending rate of flow i. This suggests either $q_l = 0$ (i.e., $t_{ql} = 0$ which means the link is not congested) or $\sum_{i \in I_l} x_i = c_l$ (which means the link is congested). We use window-based congestion control, in which a window w_i bits of flow i could be stored either in node buffers as accumulation a_i $(= \sum_{l \in L_i} q_{il})$ or on transmission links as $x_i \cdot rtt_{pi}$, where rtt_{pi} is flow i's round trip propagation delay. Note $w_i = x_i \cdot rtt_i$, where rtt_i is the round trip time observed by flow i. We summarize the above analysis to get the following result:

Proposition 1: If we use accumulation a_i as a steering parameter to control flow i's congestion window size w_i, then at the steady state we have, $\forall i \in I, \forall l \in L:$
(a) $w_i = a_i + x_i \cdot rtt_{pi}$, i.e., $a_i = x_i(rtt_i - rtt_{pi}) = x_i \cdot \sum_{l \in L_i} t_{ql}$;
(b) $t_{ql} \cdot (c_l - \sum_{i \in I_l} x_i) = 0$;
(c) $\sum_{i \in I_l} x_i \leq c_l$;
(d) $t_{ql} \geq 0$;
(e) $x_i > 0$.

Alternatively, network resource allocation can also be modeled as an optimization problem [18] [23] [20], where the system tries to maximize the sum of all flows' utility functions $\sum_{i \in I} U_i(x_i)$, in which flow i's concave utility function $U_i(x_i)$ is a measure of its welfare when it sends at a rate of $x_i > 0$, subject to a set of capacity constraints $\sum_{i \in I_l} x_i \leq c_l$ at all the links. Using the Lagrange multiplier method [6], we construct a Lagrange function $L(x, p) = \sum_{i \in I} U_i(x_i) + \sum_{l \in L} p_l \cdot (c_l - \sum_{i \in I_l} x_i)$. If utility functions are defined as $U_i(x_i) = s_i \, ln x_i$, where $s_i > 0$ is a weight, then because of the strict concavity of the objective function constrained by a convex set, the Karush-Kuhn-Tucker condition can be applied to obtain:

Proposition 2: The nonlinear programming problem

$$\begin{aligned} &maximize && \sum_{i \in I} s_i \, ln x_i && (13) \\ &subject\ to && \sum_{i \in I_l} x_i \leq c_l, \forall l \in L \\ & && x_i > 0, \forall i \in I \end{aligned}$$

has a unique global maximum. The sufficient and necessary condition for the maximum is, $\forall i \in I, \forall l \in L:$
(a) $\partial L(x,p)/\partial x_i = 0$, i.e., $s_i = x_i \cdot \sum_{l \in L_i} p_l$;
(b) $p_l \cdot (c_l - \sum_{i \in I_l} x_i) = 0$;
(c) $\sum_{i \in I_l} x_i \leq c_l$;
(d) $p_l \geq 0$;
(e) $x_i > 0$.

Now let's compare the above two results. If replacing s_i with a_i, p_l with t_{ql}, we find that Proposition 2 is turned into Proposition 1, and vice versa. This observation indicates that, by using accumulation as the steering parameter to control flow rate, the network is actually doing a nonlinear optimization in which flow i's utility function is

$$U_i(x_i) = a_i \, ln x_i. \tag{14}$$

So accumulation a_i is an instance of the weight s_i, which could be used to provide a weighted proportionally fair congestion control. Besides, the Lagrange multiplier p_l is a measure of congestion, or price explored in [23], at link l. In particular, the queuing delay t_{ql} is an instance of such price. The more severe the congestion at link l, the higher the price p_l, the larger the queuing delay t_{ql}. If there is no congestion at that link, then there is no queuing delay at all, i.e., $t_{ql} = 0$, the price p_l is also 0.

Given the above utility function of ACC, it is straightforward to show that its equilibrium bandwidth allocation is weighted proportionally fair where the accumulation a_i is the weight for flow i.