Accumulation-based Congestion Control

Yong Xia, David Harrison†, Shivkumar Kalyanaraman,
Kishore Ramachandran, Arvind Venkatesan†
ECSE and CS† Departments
Rensselaer Polytechnic Institute, Troy, NY 12180, USA
Contact email: shivkuma@ecse.rpi.edu

Abstract

This paper generalizes the TCP Vegas congestion avoidance mechanism and proposes a model to use *accumulation*, buffered packets of a flow inside network routers, as a congestion measure on which a *family* of congestion control schemes can be derived. We call this model accumulation-based congestion control (ACC). We use a fluid analysis to define accumulation and develop a general control algorithm which includes a *set* of control policies. Then we prove its proportional fairness and global stability. The ACC model serves as a reference for packet network implementations. We show that TCP Vegas is one possible scheme which fits into the ACC model. It is well known that Vegas suffers from round trip propagation delay estimation error and reverse path queuing delay. We therefore design a new scheme called Monaco which, with two FIFO queues at each router, solves these problems by employing an *out-of-band receiver-based* accumulation estimation. Analysis and simulation comparisons between Vegas and Monaco demonstrate the effectiveness of the Monaco accumulation estimator. We use ns-2 simulation and Linux kernel v2.2.18 implementation experiments to show that the static and dynamic performance of Monaco matches the theoretic results. Several key issues regarding the ACC model in general, such as the scalability of router buffer requirement and its possible solutions, are discussed.

I. Introduction

Much research has been conducted toward achieving stable, efficient and fair operation of packet-switching networks. TCP congestion control [11] is an end-to-end mechanism which has been critical for the stability of the Internet. It detects network congestion by inferring packet loss assumed to be caused only by congestion. As an alternative TCP implementation, Vegas [6] uses another measure called backlog, the estimated number of buffered packets inside network routers along the path, to detect network congestion.

Unfortunately Vegas has technical problems inherent to its backlog estimator. There has been a substantial body of work on these issues. For instance, Mo et al [19] (and references therein) pointed out Vegas' drawbacks of estimating round trip propagation delay incorrectly and possible persistent congestion. Low and Peterson [17] developed an optimization model for Vegas and suggested to improve Vegas performance by using an active queue management mechanism [1]. 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 the above problems and develop a systematic model to generalize the Vegas congestion avoidance mechanism. Formally, we define in the fluid model [8] [21] the backlog (hereafter we call *accumulation*) as a time-shifted, distributed sum of the queue contributions of a flow at a sequence of FIFO routers. We show that the dynamics of the flow's accumulation measured at a sequence of FIFO routers is *similar* (*not identical*, though) to the flow's queuing behavior at a single FIFO router. Flow rates can be controlled by controlling the accumulations in a distributed manner. This aspect is what motivates us to study a *family* of closed-loop congestion control schemes that are based upon accumulation estimation, instead of depending on packet loss for congestion detection. However, the schemes should be stable, robust and respond to unexpected packet losses.

We first develop the key concepts for this accumulation-based congestion control (ACC) model in Section II. An ACC model has two components: congestion estimation and congestion response. The congestion estimation component defines a congestion measure, i.e., accumulation and provides an implementation of the measure; while the congestion response component defines an increase/decrease policy for the source throttle. We apply queuing analysis [18] and nonlinear optimization theory [12] [16] [14] to demonstrate the equilibrium fairness characteristics of the ACC model

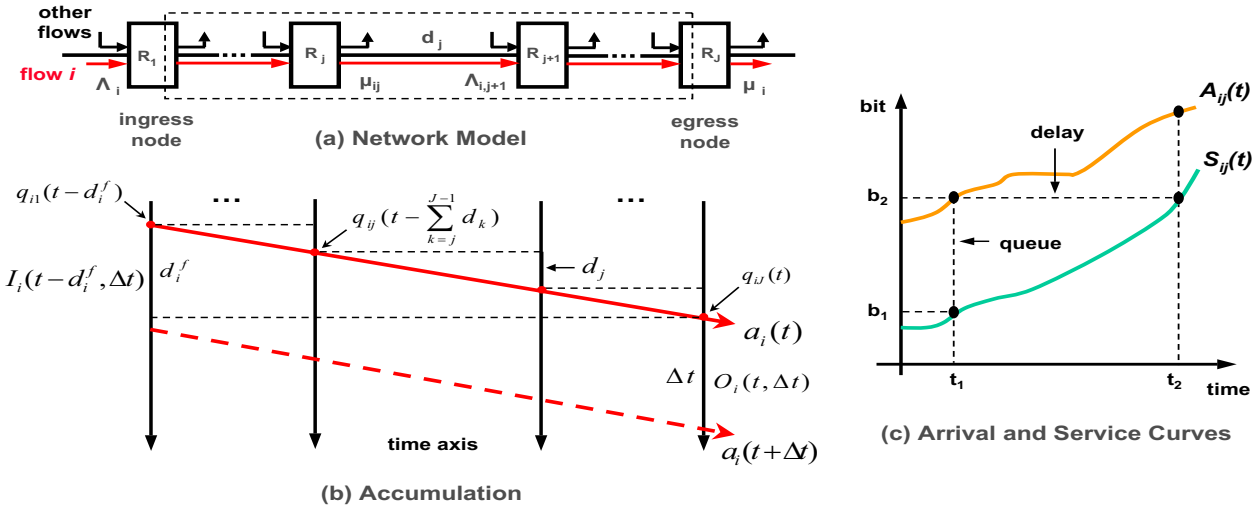


Fig. 1. Network Fluid Model

and then propose a general control algorithm which is globally stable and steers the network to the equilibrium. A range of traditional algorithms including additive-increase/additive-decrease [7] [6] and other algorithms [18] can be used in the ACC model. Detailed proofs of the fairness and stability are given in Appendix.

Within the ACC model a family of different schemes make choices in each of the ACC components and put together the entire scheme. We describe two packet network example schemes in Section III. We demonstrate that the TCP Vegas [6] congestion avoidance mechanism attempts to estimate accumulation, and fits into the ACC family. But Vegas often fails because it cannot provide an unbiased accumulation estimation. Then we develop a new scheme called Monaco that emulates the ACC fluid model in a better way. Particularly, Monaco solves the above problems of Vegas by employing an *out-of-band receiver-based* accumulation estimation. We provide resolution to a number of concerns regarding the accumulation estimation issues of Vegas in Section III-C. In Section IV we use simulations to show the static and dynamic performance of the Monaco scheme, which is also validated by a set of experiments based on a Monaco implementation in Linux OS kernel v2.2.18.

In summary, this paper develops a congestion control model using accumulation as a measure of network congestion and shows its stability and fairness characteristics. It provides a family of congestion control schemes based on this model. Vegas is an example scheme; Monaco is another one which solves some well-known problems of Vegas. Some key questions regarding the ACC model in general, such as scalability of buffer requirement, are discussed in Section V. We conclude this paper by suggesting future research issues.

II. Fluid Model

In this section we describe the ACC model. We define the accumulation concept using a bit-by-bit fluid model [8] [21] and, based on an intuition from accumulation dynamics, use accumulation to control network congestion. We show that keeping accumulation inside network routers for each flow to a target value is equivalent to a nonlinear optimization which allocates network capacity proportionally fairly. We then develop a general control algorithm for each flow to achieve its target accumulation and prove the global stability of the algorithm. Proof details are provided in Appendix.

A. Accumulation

Consider an ordered sequence of FIFO nodes $\{R_1, \ldots, R_j, R_{j+1}, \ldots, R_J\}$ along the path of a *unidirectional* flow i in Figure 1(a). The flow comes into 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[1]. At time t in any node R_j $(1 \le j \le J)$, flow i's input rate is $\lambda_{ij}(t)$, output rate

[1]In practice R_1/R_J can be mapped as source/destination to form end-to-end control loop or ingress/egress edge router to form edge-to-edge control loop. Here we focus on the ACC model itself so we'll discuss architecture issues in a separate paper.

$\mu_{ij}(t)$. The propagation delay from node R_j to node R_{j+1} is d_j.

We define the arrival curve $A_{ij}(t)$ of a flow i at a node R_j as the number of bits from that flow which have cumulatively arrived at the node up to time t, and similarly the service curve $S_{ij}(t)$ as flow i's bits cumulatively serviced at node R_j [8] [21], drawn in Figure 1(c). For any FIFO node R_j, both $A_{ij}(t)$ and $S_{ij}(t)$ are continuous[2] and non-decreasing functions. If there is no packet loss, then at any time t, by definition, flow i's buffered bits $q_{ij}(t)$ in node R_j is the difference between $A_{ij}(t)$ and $S_{ij}(t)$, as shown in Figure 1(c):

$$q_{ij}(t) = A_{ij}(t) - S_{ij}(t). \tag{1}$$

We compute the change of flow i's queued bits at node R_j

$$\begin{aligned} \Delta q_{ij}(t) &= q_{ij}(t+\Delta t) - q_{ij}(t) \\ &= [A_{ij}(t+\Delta t) - A_{ij}(t)] - [S_{ij}(t+\Delta t) - S_{ij}(t)] \\ &= [\overline{\lambda}_{ij}(t,\Delta t) - \overline{\mu}_{ij}(t,\Delta t)] \times \Delta t \\ &= I_{ij}(t,\Delta t) - O_{ij}(t,\Delta t) \end{aligned} \tag{2}$$

where $I_{ij}(t,\Delta t)$ and $O_{ij}(t,\Delta t)$ are incoming and outgoing bits of flow i at node R_j during the time interval $[t, t+\Delta t]$; $\overline{\lambda}_{ij}(t,\Delta t)$ and $\overline{\mu}_{ij}(t,\Delta t)$ are the correspondent average input and output rates, respectively.

Now consider the flow's queuing behavior at a *sequence* of FIFO nodes. Reasonably, suppose data link layer transmission could be modelled as a line, then flow i's input rate $\lambda_{i,j+1}(t)$ at a node R_{j+1} is a delayed version of its output rate $\mu_{ij}(t)$ at the upstream neighbor node R_j, namely

$$\mu_{ij}(t - d_j) = \lambda_{i,j+1}(t). \tag{3}$$

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

$$a_i(t) = \sum_{j=1}^{J} q_{ij}(t - \sum_{k=j}^{J-1} d_k) \tag{4}$$

which is shown as the solid slant line in Figure 1(b). Note this definition includes only those bits backlogged inside the node buffers, not those stored on transmission links. Using it as a reference we provide an unbiased accumulation estimator in Section III-B.1. We define flow i's ingress and egress rates as those at the ingress and egress nodes, respectively:

$$\begin{aligned} \lambda_i(t) &= \lambda_{i1}(t) \\ \mu_i(t) &= \mu_{iJ}(t). \end{aligned} \tag{5}$$

Using Equations (2)–(5), we calculate flow i's accumulation change as follows:

$$\begin{aligned} \Delta a_i(t) &= a_i(t+\Delta t) - a_i(t) \\ &= \sum_{j=1}^{J} \Delta q_{ij}(t - \sum_{k=j}^{J-1} d_k) \\ &= [\overline{\lambda}_i(t - d_i^f, \Delta t) - \overline{\mu}_i(t,\Delta t)] \times \Delta t \\ &= I_i(t - d_i^f, \Delta t) - O_i(t,\Delta t) \end{aligned} \tag{6}$$

where $d_i^f = \sum_{j=1}^{J-1} d_j$ is the forward direction propagation delay of flow i from node R_1 all the way down to node R_J. Similar to Equation (2), $I_i(t - d_i^f, \Delta t)$ and $O_i(t,\Delta t)$ are flow i's bits coming into and going out of network during two *different* time intervals but both of length Δt; while $\overline{\lambda}_i(t - d_i^f, \Delta t)$ and $\overline{\mu}_i(t,\Delta t)$ are the correspondent average ingress

[2]Strictly this is true if we accept that a bit is infinitely small.

and egress rates. The result, illustrated in Figure 1(b), shows the change of a flow's accumulation on its path is only related to its input and output at the ingress and egress nodes. Further, this means it is possible to control accumulation at only the ingress and egress nodes.

For one FIFO node, it's straight-forward to control flow rates by controlling the number of queued packets [10], since packets buffered decide service received if the scheduling discipline is FIFO. Due to the similarity of Equations (2) and (6), for a sequence of FIFO nodes, we aim to *control flow rates by controlling the accumulations, i.e., keeping a steady state accumulation for each flow* inside network. Note Equations (2) and (6) have a significant difference of the propagation delay d_i^f, which is a constant as long as flow i's route is fixed.

B. Queuing and Optimization Analysis

To facilitate better understanding of using accumulation as the steering parameter for congestion control, we provide physically a simple queuing analysis and mathematically an optimization theory to demonstrate the *steady state* picture by leveraging results from [12] [16] [18]. It turns out that the ACC model steers the network to an equilibrium of proportionally fair bandwidth allocation. Then we develop a globally stable control algorithm to drive the network to the equilibrium.

Network congestion control can be formalized as a resource allocation problem [12] [16] [14] [18]. 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 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 $I_l = \{i \in I \mid i \text{ traverses } l\}$.

Let's firstly consider from queuing system perspective [18]. After the system approaches a steady state (so we neglect the time variable t in the last subsection), 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 $\sum_{i \in I_l} x_i$. This suggests either $q_l = 0$ (i.e., $t_{ql} = 0$) or $\sum_{i \in I_l} x_i = c_l$. We use window-based flow 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 $x_i \cdot rtt_{p\,i}$, where x_i is the flow rate and $rtt_{p\,i}$ is its round trip propagation delay. Observing that $w_i = x_i \cdot rtt_i$, we summarize to get the following

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 (achievable by the control algorithm in Section II-C) we have, $\forall i \in I, \forall l \in L$:
(a) $w_i = a_i + x_i \cdot rtt_{p\,i} \Rightarrow a_i = x_i(rtt_i - rtt_{p\,i}) = x_i \cdot \sum_{l \in L_i} t_{ql}$;
(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 modelled as a nonlinear optimization problem [12] [16] [14]. The network tries to maximize the sum of all flows' utility functions $\sum_{i \in I} U_i(x_i)$, in which flow i's utility function $U_i(x_i)$ is a measure of its happiness 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 links. Using Lagrange multiplier method, we construct a Lagrange function $L(\vec{x}, \vec{p}) = \sum_{i \in I} U_i(x_i) + \sum_{l \in L} p_l \cdot (c_l - \sum_{i \in I_l} x_i)$. If $U_i(x_i)$ is 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, we apply the Karush-Kuhn-Tucker condition [3] to obtain

Proposition 2: The nonlinear programming problem

$$\begin{aligned} maximize \quad & \sum_{i \in I} s_i \, ln x_i \\ subject\ to \quad & \sum_{i \in I_l} x_i \leq c_l, \forall l \in L \\ & x_i > 0, \forall i \in I \end{aligned} \tag{7}$$

has a unique global maximum. The sufficient and necessary condition for an $\vec{x}$ to be the maximum is, $\forall i \in I, \forall l \in L$:
(a) $\partial L(\vec{x}, \vec{p})/\partial \vec{x} = 0 \Rightarrow s_i = x_i \cdot \sum_{l \in L_i} p_l$;
(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 propositions. 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 a steering parameter to control flow rates, the network is actually doing a nonlinear optimization in which flow i's utility function is

$$U_i(x_i) = a_i \, ln x_i. \tag{8}$$

It turns out that accumulation a_i is an instance of the weight s_i, which could be used to provide a *weighted* proportionally fair flow control as proved in Section II-D and Appendix. Besides, the Lagrange multiplier p_l is a measure of congestion, or price explored in [16], 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, i.e., $t_{ql} = 0$, the price p_l is also 0.

C. Control Algorithm

In the ACC model we use accumulation to measure network congestion as well as to probe available bandwidth. If accumulation is low, we increase congestion window; otherwise, we decrease it to drain accumulation. Specifically, we try to maintain a constant target accumulation a_i^* for each flow i by applying a general ACC control algorithm:

$$\dot{w}_i(t) = -\kappa \cdot f(a_i(t) - a_i^*) \tag{9}$$

where $w_i(t),\ a_i(t)$ and a_i^* are respectively the congestion window size, accumulation and target accumulation value of flow i, $f(\cdot)$ is an increasing function with a unique root 0 (i.e., only $f(0) = 0$) and $\forall\, y \neq 0,\ yf(y) > 0$, and $\kappa > 0$.

Obviously Equation (9) includes a *set* of algorithms. The reason we present a general algorithm here is that *all instance algorithms which fit into Equation (9) share a common steady state property of proportional fairness*, as shown in the next subsection and Appendix, where we also prove that the above algorithm is globally stable.

By choosing different f functions, we instantiate the above general algorithm into a set of control policies including the well-known additive-increase-additive-decrease (AIAD) policy [7] popularized by TCP Vegas [7], an algorithm proposed by Mo and Walrand [18], and another proportional control policy, among others. We list only these three policies below.

i) ACC-AIAD is obtained by choosing f as a threshold function. It additively increases and additively decreases the window value:

$$\dot{w}_i(t) = -\frac{\kappa}{rtt} \cdot sgn(a_i(t) - a_i^*) \tag{10}$$

where $sgn(\cdot)$ is the sign function defined by

$$sgn(y) = \begin{cases} +1 & if \quad y > 0 \\ 0 & if \quad y = 0 \\ -1 & if \quad y < 0. \end{cases} \tag{11}$$

ii) ACC-MW is proposed by Mo and Walrand [18]:

$$\dot{w}_i(t) = -\kappa \cdot \frac{rtt_p}{rtt} \cdot \frac{a_i(t) - a_i^*}{w_i(t)}. \tag{12}$$

iii) ACC-P is generated by choosing f as a linear identity function. It is a proportional control policy:

$$\dot{w}_i(t) = -\frac{\kappa}{rtt} \cdot (a_i(t) - a_i^*). \tag{13}$$

Due to the space limit, we don't list other policies here. We simulated all the above control policies and verified that they do achieve the same steady state bandwidth allocation.

D. Properties

For any flow control algorithm, major theoretic concerns are its stability, fairness and queue bound. Stability is to guarantee equilibrium operation of the algorithm. Fairness, e.g., max-min [4] and proportional fairness [12], determines the allocation of network bandwidth among competing flows. Queue bound provides an upper limit on the router buffer requirement, which is important for real deployment. We prove in Appendix that

Proposition 3: The flow control algorithm given by Equation (9) is globally stable and weighted proportionally fair.

Even we keep a finite accumulation inside network for every flow, the steady state queue at a node scales up to the number of flows passing that bottleneck. In practice, we need to provide more buffers to avoid packet loss and make the control algorithm robust to such loss (see Section III-B). Another way to alleviate this problem is to control aggregate flow, instead of individual source-destination flow. More details on buffer size scalability are discussed in Section V.

Interestingly, as Proposition 3 states, different ACC control policies can achieve the same fairness property, as long as they fit into Equation (9). Thus to achieve a particular steady state performance, we have the freedom to choose from a set of control policies. In this sense, we regard *ACC flow control as a two-step issue of setting a target steady state allocation (fairness) and then designing a control policy (stability and dynamics) to achieve that allocation*.

III. ACC Schemes

In this section we instantiate the ACC fluid model into two example schemes for packet-switching networks. Firstly we show that TCP Vegas [6] tries to estimate accumulation and fits into the ACC model. Unfortunately Vegas often fails to provide an unbiased accumulation estimation. Then we design a new scheme called Monaco which solves the estimation problems of Vegas. Monaco also improves the congestion response by utilizing the value of estimated accumulation, unlike Vegas' AIAD policy which is possibly slow in reacting a sudden change in demands or network capacity. By comparing Monaco and Vegas via analysis and simulation we reach two observations: It is effective to employ 1) a *receiver-based* mechanism and, 2) the measurement of *forward path queuing delay*, instead of round trip queuing delay as in Vegas, to estimate accumulation unbiasedly. The scheme design is guided by the following goals:

Goal 1: **Stability:** The scheme should converge to 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 converge 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 [6] was proposed as an alternative TCP implementation. It includes several modifications over TCP Reno [11]. However, we focus only on its congestion avoidance mechanism, which fits well as an example ACC scheme.

The Vegas estimator for accumulation was originally called "backlog", a term we use interchangeably in this paper. For each flow, the Vegas estimator takes as input an estimate of its round trip propagation delay, hereafter called rtt_p (or $basertt$ in [6] [19]). Vegas then estimates the backlog as

$$\begin{aligned} \hat{a}_V &= (expected\,rate - actual\,rate) \times rtt_p \\ &= (\frac{cwnd}{rtt_p} - \frac{cwnd}{rtt}) \times rtt_p \end{aligned} \tag{14}$$

which could be simplified as

$$\hat{a}_V = \frac{cwnd}{rtt} \times rtt_q \tag{15}$$

where $cwnd/rtt$ is the average sending rate during that round trip time (RTT) and $rtt_q = rtt - rtt_p$ is the round trip queuing delay. According to Little's Law, $\hat{a}_V$ is the backlogged packets inside bottleneck routers. If rtt_p is accurately available and there is no reverse path queuing delay, then $\hat{a}_V$ provides an unbiased estimation for accumulation.

Vegas estimates the rtt_p as the minimum RTT measured so far. So, if the queues drain often, it is likely that each control loop will eventually obtain a sample that reflects the rtt_p. 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 (10), i.e.,

$$cwnd(n+1) = \begin{cases} cwnd(n) + 1 & if \quad \hat{a}_V < \varepsilon_1 \\ cwnd(n) - 1 & if \quad \hat{a}_V > \varepsilon_2 \end{cases} \tag{16}$$

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 rtt_p . Vegas misinterprets such an increase as less congestion and sends faster. Hence, this policy can lead to unbounded queue which introduces persistent congestion and loss [17], violating Goal 1 and 4. Mo et al [19] 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. But we show in Section III-C that since Vegas uses an AIAD policy, it cannot guarantee queue drain at intermediate bottlenecks within k RTTs.

• *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 (4) 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 firstly the Monaco accumulation estimator and then its congestion response policy.

B.1 Monaco: Congestion Estimation Protocol

According to its definition, accumulation of a flow is the sum of its queued bits at a sequence of FIFO routers, including both ingress and egress nodes as well as intermediate routers. We aim to eliminate the computation at intermediate routers to keep them simple. Actually it is impossible for *all* nodes R_j $(1 \leq j \leq J)$ to compute synchronously their queues $q_{ij}(t - \sum_{k=j}^{J-1} d_k)$ since no common clock is maintained.

To estimate accumulation without explicit computation at intermediate routers, Monaco generates a pair of back-to-back control packets once per RTT at the ingress node as shown in Figure 2. 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. 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 current queueing 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 flow rates at all routers do not change dramatically, then by Little's Law, the number of data packet arrivals at the egress node after the OB control packet, but before the IB control packet equals the accumulation. In Figure 2, 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.

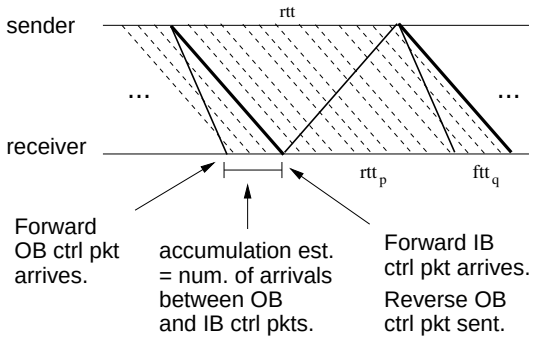


Fig. 2. Monaco Accumulation Estimator

In a real packet network there are some constraints which might introduce systematic accumulation estimation errors. For instance, beyond propagation and queuing delays, a packet also experiences switching and transmission delays not considered in the fluid model. But since both OB and IB control packets are of the same size (32 bytes) and sent out back-to-back from the ingress node, their switching and transmission delays cancel out. Another aspect is non-preemptive packet forwarding. When an OB control packet arrives at a router, it has to wait if there is another packet being serviced. So the delay experienced by the OB control packet is generally not constant and larger than the propagation delay d_i^f, so the measured time interval ftt_q seems less than its true value. But as shown in Figure 2, if we just *count* the number $\hat{a}_M$ of data packet arrivals in that interval, we can at least alleviate the effect of noises included in the time interval ftt_q, as long as priority queue and FIFO properties are true.

Besides, we need to consider the effect of traffic burstiness. When we have a congestion window size $cwnd$, we also compute a rate based RTT estimation: $rate = cwnd/rtt$. We use this rate value to smooth incoming traffic and thus alleviate the effect of burstiness.

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 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 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: congestion feedback, i.e., 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 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 [11]. These routine reliability enhancements are similar to the Congestion Manager protocol [2].

B.2 Monaco: Congestion Response Protocol

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

For Monaco we choose a window-based instead of rate-based control policy because the former is more conservative and rate is hard to accurately measure in practice. Since pure window-based control policy might introduce burstiness we use *rate-modulated window* control to smooth incoming traffic into packet networks by employing a token bucket shaper with a rate value of $cwnd/rtt$.

We provide below the Monaco's proportional control policy[3] which is the discrete version of Equation (13):

$$cwnd(n+1) = cwnd(n) - \kappa \cdot (\hat{a}_M - a^*) \tag{17}$$

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.

[3] Actually Monaco can choose from a set of control policies, as discussed in Section II-C. We show only one here.

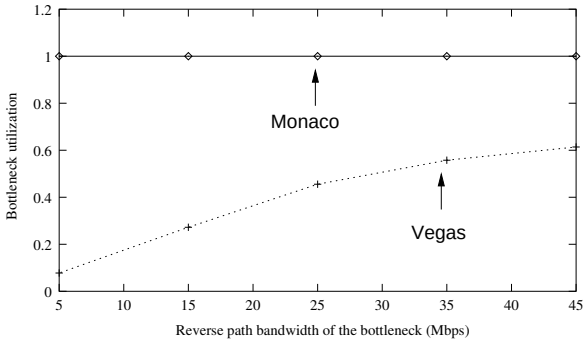
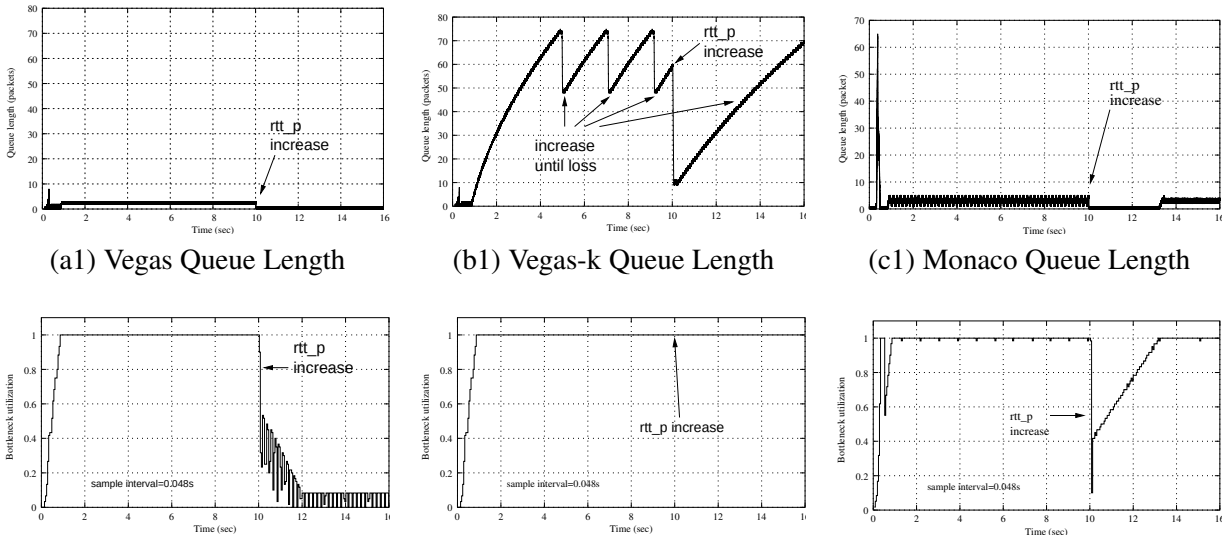


Fig. 3. Comparison between Vegas and Monaco under Reverse Path Congestion



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

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

Fig. 4. Comparison between Vegas, Vegas-k and Monaco under rtt_p(or $basertt$) Estimation Error

Monaco improves Vegas' control policy by utilizing the value of accumulation estimation fedback by the reverse OB control packet, instead of regarding it as a 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 size as in TCP Reno.

C. Comparisons of Vegas and Monaco

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

Vegas estimator operates at *sender* side. According to Equation (15) it actually calculates:

$$\hat{a}_V = \frac{cwnd}{rtt} \times (rtt - rtt_p) \tag{18}$$

$$= \frac{cwnd}{rtt} \times (t_q^f + t_q^b) \tag{19}$$

where t_q^f and t_q^b are forward and reverse direction queuing delays, respectively. The above equations imply that Vegas may suffer from two problems: 1) By Equation (19), if there exists reverse direction 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 there is no control over reverse direction flows. To show this effect we use a simple dumb-bell topology with

TABLE I
EXPERIMENT PARAMETERS

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

the bottleneck of 45Mbps forward direction bandwidth shared by 7 forward direction flows and 7 reverse flows. We change the bottleneck's reverse direction bandwidth from 5Mbps to 45Mbps. As shown in Figure 3, Vegas utilization is only $10\% \sim 60\%$. 2) By Equation (18), if rtt_p is overestimated, then Vegas underestimates accumulation. This leads to extra steady queue in bottlenecks or even persistent congestion. Results for a single bottleneck of 10Mbps bandwidth and 12ms delay which is used by one flow employing Vegas and Vegas-k are shown in Figures 4(a) and 4(b), where rtt_p estimation error is introduced by a sudden rtt_p change to 52ms at time 10s. Vegas operates with very low utilization of less than 10% and Vegas-k operates with queue increase until loss occurs.

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, 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 be impossible with in-band control packets. In summary, the sensitivity issues with Vegas point to a *fundamental* problem with the in-band techniques for accumulation estimation.

Monaco solves both problems. Monaco estimator operates at *receiver* side and thus excludes the effect of reverse path congestion. By counting the data packets arriving between in- and out-of-band control packets, Monaco does not explicitly need to estimate the forward direction propagation delay d_i^f. (Actually the out-of-band control packets provide implicitly this value.) More specifically, since Monaco implements a rate-paced window control algorithm to smooth out incoming traffic, the time difference between the in- and out-of-band control packets gives a sample of the current forward direction 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 out-of-band control packet also makes Monaco adaptive to re-routing since it is sent every RTT. As shown in Figure 3 and Figure 4(c), after a brief transient period, Monaco operates 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 (including Vegas-k) and Monaco suggest two important observations on how to estimate accumulation unbiasedly: 1) The key is to measure *forward direction queuing delay* (via out-of- and in-band 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 get rid of the effect of reverse path queuing delay which is hardly under forward direction congestion control.

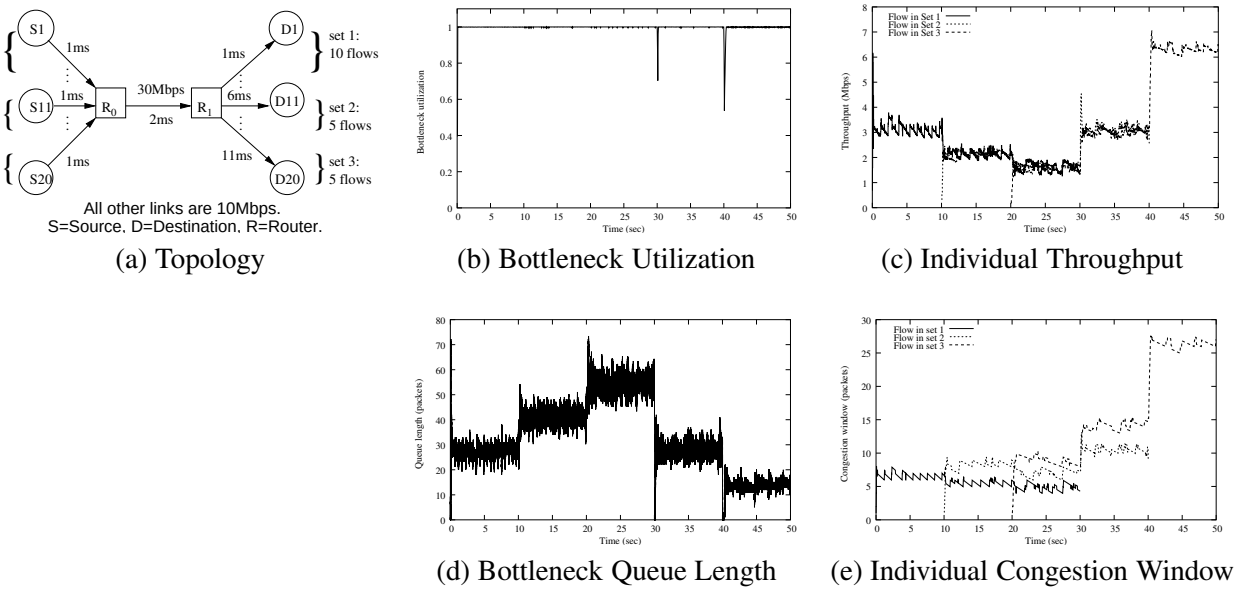
IV. SIMULATION AND IMPLEMENTATION EXPERIMENTS

In the last section we have shown that Monaco performs better than Vegas, so we focus on evaluating Monaco scheme by ns-2 simulation and Linux implementation in this section. Our experiments illustrate:

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

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

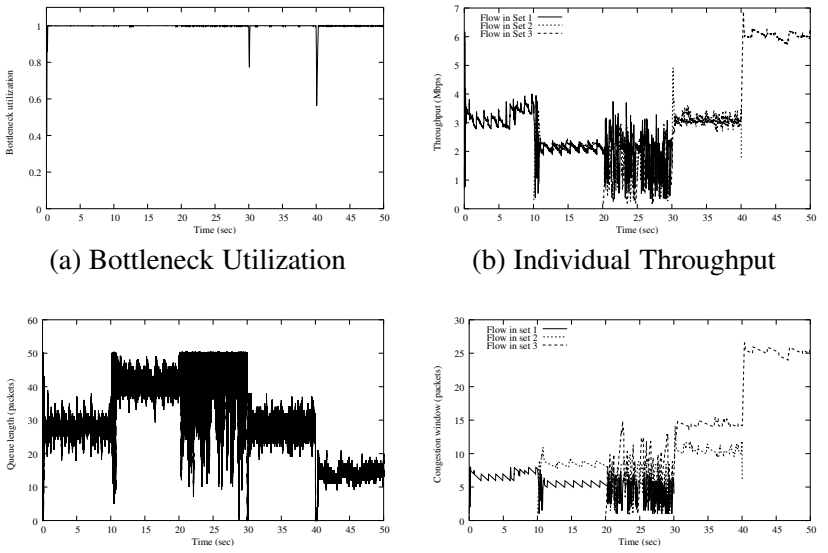
In all simulation experiments we use ns-2 simulator [20] with parameters given in Table I. We also implement the Monaco scheme in Linux OS (kernel version 2.2.18) based on the Click configurable router [13]. Experimental results from implementation match with those of simulations. In brief, in combination with Section III-C, this section shows that the Monaco scheme satisfies all the goals outlined in Section III.



(a) Topology (b) Bottleneck Utilization (c) Individual Throughput

(d) Bottleneck Queue Length (e) Individual Congestion Window

Fig. 5. Monaco with Enough Buffer in Bottleneck



(a) Bottleneck Utilization (b) Individual Throughput

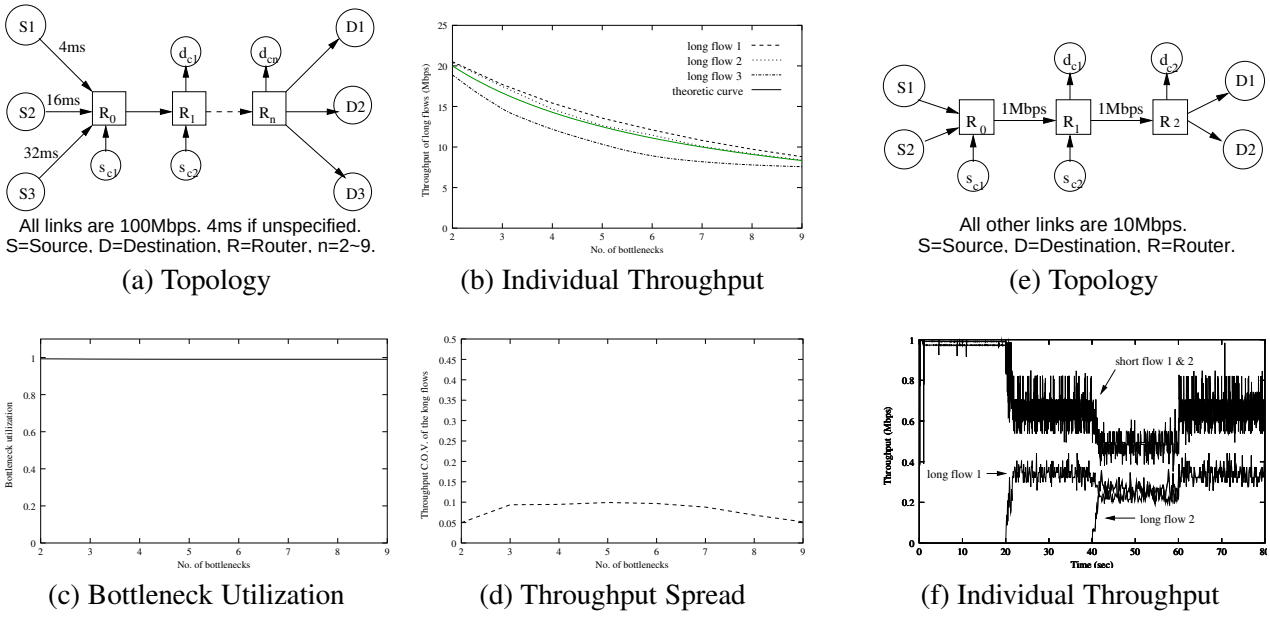
(c) Bottleneck Queue Length (d) Individual Congestion Window

Fig. 6. Monaco with Underprovisioned Buffer in Bottleneck

A. Single Bottleneck with Dynamic Demands

Firstly we consider a single 30Mbps bottleneck with 2ms propagation delay shared by 3 sets of flows using the Monaco scheme. The topology is shown in Figure 5(a). Set 1 has 10 flows starting at 0s and stopping at 30s; Set 2 has 5 flows starting at 10s and stopping at 40s; Set 3 has 5 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 3 sets of flows are 4ms, 9ms and 14ms, respectively. We simulate for 50s. We performed two simulations, one with enough buffer provided for the bottleneck, the other with underprovisioned buffer.

In the first simulation, the bottleneck router has enough buffer of 90 packets, as shown in Figure 5(d), there is no packet loss. We randomly pick one flow from each set and draw its individual throughput in Figure 5(c). We observe that from 0s to 30s, the throughput is about 3Mbps, since only 10 flows are active; When the 5 flows from set 2 jump in at 10s, the throughput drops to 2Mbps, as we have 15 active flows. Similarly, when the final 5 flows from set 3



(a) Topology (b) Individual Throughput (e) Topology

(c) Bottleneck Utilization (d) Throughput Spread (f) Individual Throughput

Fig. 7. Monaco Fairness via ns-2 Simulation (a-d) and Linux Implementation (e,f)

enter at 20s, the throughput changes to 1.5Mbps. Then at 30s, the 10 flows of set 1 stop, the throughput increases to 3Mbps. At 40s, the 5 flows of set 2 leave, only the 5 flows of set 3 are in the system with throughput of about 6Mbps. The congestion window dynamics is similar as shown in Figure 5(e). Bottleneck queue length is depicted in Figure 5(d) where incoming flows build up a steady queue and 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 bottleneck utilization always stays around 100%, except two soon-recovered drops during abrupt demand changes at 30s and 40s as seen in Figure 5(b). From this simulation, we validate that Monaco demonstrates a stable behavior under a dynamic and heterogeneous environment and keeps a steady queue inside bottleneck.

In the second simulation, the bottleneck router buffer is underprovisioned, as illustrated in Figure 6(c), we can see that the queue length grows to the limit of the whole buffer size of 50 packets, and there is a correspondent packet loss leading to halving of the congestion window during 20s $\sim$ 30s in Figure 6(d). Consequently, throughput is more oscillating as in Figure 6(b), but the bottleneck is still fully utilized. From this simulation, we see that without enough buffer, Monaco shows a degraded behavior under dynamically changing demands.

B. Multiple Bottlenecks

Now we show the steady state performance of Monaco when flow traverses more than one bottleneck. We use a linear topology with multiple congested links depicted in Figure 7(a). We did a set of simulation experiments by changing the number of bottlenecks N from 2 to 9. There are 3 "long" flows passing all bottlenecks and a set of "short" flows each using only one bottleneck. Every link has 100Mbps capacity and 4ms delay. The long flows have very different RTTs. We simulated under only one condition with enough buffer provided for the routers. As already shown in the last subsection, if buffer is not enough, the Monaco scheme degrades. For the buffer requirement scalability problem, we discuss further in the next section.

As illustrated in Figure 7(b), the throughput curve for each individual long flow is located right around the theoretic one of $100/(3+N)$Mbps. So each 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 7(d), which is about 5% $\sim$ 10%. The bottleneck utilization for $R0 - R1$ link is shown in Figure 7(c), which is always 100% during the whole simulation of 60s.

We also did a set of experiments using our Linux implementation of Monaco. Here we show one with dynamic demands. For more details, please refer to [22]. We have 2 bottlenecks each of 1Mbps capacity as drawn in Figure 7(e). During the 80s of 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 7(f). For instance, the first long flows' throughput starts with 0.33Mbps (its fair share) at 20s and changes to some 0.25Mbps at 40s when the second long flow shows up. After that, the second long flow gets about its fair share of 0.33Mbps.

These simulation and implementation experiments demonstrate that, with enough buffer provisioned, Monaco achieves a proportionally fair bandwidth allocation in a multiple bottleneck case, validating our theoretic results of Proposition 2.

V. Summary and Future Work

In this paper we generalize TCP Vegas and develop a congestion control model using accumulation which is buffered packets of a flow inside network routers as a measure to detect and control network congestion. By applying a simple queuing analysis and nonlinear optimization theory on the fluid model, we prove that the ACC model allocates network bandwidth proportionally fairly at the equilibrium – which is its steady state feature. We propose a set of globally stable control algorithm which drive the network toward that equilibrium – which is related to its dynamic characteristics. A family of schemes, including Vegas, could be derived based on the ACC model. Using the model as a reference, we design a new example scheme Monaco which, with two priority FIFO queues provided by network routers, solves the well-known problems of Vegas. In particular, through analysis and simulation, we show a receiver-based out-of-band accumulation estimator is effective to produce an unbiased accumulation measurement.

We use a set of simulations to evaluate the dynamic and steady state performance of Monaco under different topologies and conditions. Specifically, the Monaco scheme demonstrates it's effectiveness in keeping system stable, fair, and highly utilized, given enough buffers in the bottleneck routers. With underprovisioned buffer, Monaco's performance is degraded. We also implement the Monaco scheme in Linux OS kernel v2.2.18 based on the Click modular router [13] and validate most of the simulation results in an internal testbed network. In summary, the main contributions of this paper are:

- a mathematically defined, physically meaningful concept of backlogged packets – accumulation as a measure of network congestion;

- a model of accumulation-based congestion control with provable global stability and proportional fairness based on which a family of schemes could be derived;

- a comparison between Vegas and Monaco to show that Monaco's receiver-based out-of-band accumulation measurement solves Vegas' well-known estimation problems;

- a Monaco scheme implemented as a packet network protocol which estimates accumulation unbiasedly and utilizes this value in a non-binary manner to control congestion.

One critical question that remains is the scalability of the router buffer requirement which is proportional to the number of flows passing that router. Since the ACC model can be mapped end-to-end or edge-to-edge (though we focus on the model itself and don't elaborate the architecture issues in this paper), one way to alleviate this problem is to control aggregate flow in an edge-to-edge manner, instead of end-to-end for each micro-flow of source-destination pair. A possibly better solution to keep buffer size bounded is to use an appropriate queue management mechanism [5] [9] such as virtual queuing [15]. Further, by keeping different accumulation for different flows, it's possible to provide service differentiation. These issues are being explored in our ongoing research.

VI. Acknowledgments

This work was supported in part by National Science Foundation under contracts ANI-9806660 and ANI-9819112 and a grant from Intel Corp. The authors also thank Sthanuthan Ramakrishnan for his related work and Xinzhe Fan for discussion on the proof of Theorem 1 in Appendix.

REFERENCES

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

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

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

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

[5] B. Braden et al. Recommendations on Queue Management and Congestion Avoidance in the Internet. *IETF RFC 2309*, Apr 1998.

[6] 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, Oct 1995.

[7] 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.

[8] R. Cruz. Quality of Service Guarantees in Virtual Circuit Switched Networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1048-1056, Aug 1995.

[9] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397-413, Aug 1993.

[10] R. Guérin, S. Kamat, V. Peris and R. Rajan. Scalable QoS Provision Through Buffer Management. In *Proc. SIGCOMM'98*, Sept 1998.

[11] V. Jacobson. Congestion Avoidance and Control. In *Proc. SIGCOMM'88*, Aug 1988.

[12] 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.

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

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

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

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

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

[18] J. Mo and J. Walrand. Fair End-to-End Window -based Congestion Control. *IEEE/ACM Trans. on Networking*, 8(5):556-567, Oct 2000.

[19] J. Mo, R. La, V. Anantharam and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *Proc. INFOCOM'99*, Mar 1999.

[20] Network Simulator ns-2. Http://www.isi.edu/nsnam/ns/.

[21] A. Parekh and R. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Trans. on Networking*, 1(3):344–357, Jun 1993.

[22] A. Venkatesan. An Implementation of Accumulation-based Congestion Control Schemes. *RPI-CS M.S. Thesis*, In Preparation, 2002.

APPENDIX

In this section we prove briefly the stability of the general fluid model control algorithm[4] of Equation (9) by showing its Lyapunov function is strictly decreasing in time and the fairness of the ACC model using the nonlinear optimization model defined by Equation (7). The proof of stability is based on a result obtained by Mo and Walrand in [18].

A. Stability

For conciseness we rewrite in matrix form the queuing analysis provided in Section II-B. Denote sending rate $x = [x_1, ..., x_I]^T$, $X = diag(x)$, congestion window $w = [w_1, ..., w_I]^T$, round trip propagation delay $d = [rtt_{p1}, ..., rtt_{pI}]^T$, $D = diag(d)$, round trip time $\bar{d} = [rtt_1, ..., rtt_I]^T$, $\bar{D} = diag(\bar{d})$, link capacity $c = [c_1, ..., c_L]^T$, queuing time $q = [t_{q1}, ..., t_{qL}]^T$, $Q = diag(q)$, and binary routing matrix $A = [A_{il}, i \in I, l \in L]$ where $A_{il} = 1$ if flow i traverses link l, otherwise $A_{il} = 0$. Then the steady state queuing condition of Proposition 1 is $w = X(Aq+d)$, $Q(A^T x - c) = 0$, $A^T x \leq c$, $q \geq 0$, and $x > 0$.

According to [18], the routing matrix A can be downsized to a full-rank matrix A_B which includes only bottlenecks where $A_B^T x = c_B$. This is intuitively true because if there exists a link l which is not congested, i.e., $\sum_{i \in I_l} x_i < c_l$, then that link can be deleted from the above equations since $t_{ql} = 0$. Further, the mapping between congestion window vector w and rate vector x is unique (see Theorem 2 of [18]) and we have the following

[4] Obviously in the proof of stability we don't consider the effect of round trip delay which is an issue for future research.

Lemma 1 (Lemma 4 of [18]): The Jacobian $J_x = dx/dw = [\partial x_i/\partial w_j, i, j \in I]$ is given by

$$J_x = \bar{D}^{-1}(\mathbf{1} - Z) \tag{20}$$

where

$$Z = XA_B(A_B^T X \bar{D}^{-1} A_B)^{-1} A_B^T \bar{D}^{-1} \geq 0. \tag{21}$$

Proof. Please refer to Appendix B of [18] for details.

Lemma 2: The Jacobian $J_f = df/dw = [\partial f_i/\partial w_j, i, j \in I]$ is positive definite, where $f_i = f(\, a_i(t) - a_i^*\,)$.
Proof. Note $a_i(t) = w_i(t) - x_i(t) \cdot rtt_{pi}$, thus

$$f_i = f(\, w_i(t) - x_i(t) \cdot rtt_{pi} - a_i^*\,). \tag{22}$$

If denote $f_i' = f'(a_i(t) - a_i^*)$, then

$$\frac{\partial f_i}{\partial w_j} = f_i' \cdot [\, \frac{\partial w_i}{\partial w_j} - rtt_{pi}\frac{\partial x_i}{\partial w_j}\,]. \tag{23}$$

Note $f(\cdot)$ is strictly increasing, so $f_i' > 0$. If denote $F' = diag[f_i', i \in I]$, according to *Lemma 1*, we have

$$\begin{aligned} J_f &= F' \cdot [\mathbf{1} - DJ_x] \\ &= F' \cdot [\mathbf{1} - D\bar{D}^{-1} + D\bar{D}^{-1}Z] \\ &> 0. \end{aligned} \tag{24}$$

Theorem 1: $V(w) = \sum_{i\in I}[f(\, a_i(t) - a_i^*\,)]^2$ is a Lyapunov function for the system (9) which is globally stable.
Proof. Since $f(\cdot)$ has a unique root at 0, obviously $V(w) = 0$ only if $a_i(t) = a_i^*, \forall i$; otherwise $V(w) > 0$. If we denote $F = [f_1, ..., f_I]^T$, then according to *Lemma 2* we get

$$\begin{aligned} \frac{dV(w)}{dt} &= \sum_{j\in I}\frac{\partial V}{\partial w_j}\cdot\frac{dw_j}{dt} \\ &= \sum_{j\in I}\sum_{i\in I}\frac{\partial V}{\partial f_i}\cdot\frac{df_i}{dw_j}\cdot\frac{dw_j}{dt} \\ &= -2\kappa\sum_{j\in I}\sum_{i\in I} f_i\frac{df_i}{dw_j}f_j \\ &= -2\kappa F^T J_f F \\ &< 0. \end{aligned}$$

B. Fairness

Theorem 2: Equilibrium rate allocation of the system (9) is weighted proportionally fair [12].
Proof. The equilibrium of Equation (9) is characterized by $a_i(t) = a_i^*$, for every flow i. According to the equality of Propositions 1 and 2 in Section II-B, since Monaco is doing the nonlinear optimization problem (7) which is constrained by a convex set, and flow i's utility function $U_i(x_i) = a_i lnx_i$ is strictly concave, according to the optimality condition [3], if x_i^* is the equilibrium allocation to flow i, then for any other feasible allocation x_i, we have

$$\sum_{i\in I} U_i'(x_i^*)\cdot(x_i - x_i^*) = \sum_{i\in I} a_i^*\cdot\frac{x_i - x_i^*}{x_i^*} < 0 \tag{25}$$

which is the definition of weighted proportional fairness.