An Accumulation-based Congestion Control Model

Yong Xia, David Harrison†, Shivkumar Kalyanaraman, Kishore Ramachandran, Arvind Venkatesan†
ECSE and CS† Departments, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

***Abstract*— This paper[1] 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 bit-by-bit fluid model to define the accumulation concept 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 solves these problems by employing an *out-of-band receiver-based* accumulation estimator, with minimal support from network routers. Analysis and simulation comparisons between Vegas and Monaco demonstrate the effectiveness of the Monaco accumulation estimator. We use ns-2 simulations 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 router buffer requirement, is discussed.**

I. INTRODUCTION

Much research has been conducted toward achieving stable, efficient and fair operation of packet-switching networks. TCP congestion control [7] 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 [3] uses another measure called backlog, the number of buffered packets inside network, to detect network congestion. Unfortunately Vegas has technical problems inherent to its backlog estimator which prevent it from functioning properly. There has been a substantial body of work on these issues, such as [1] [13] [12]. 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 the backlog (hereafter we call *accumulation*) as a time-shifted, distributed sum of the queue contributions of a flow at a set of FIFO routers on its path. We show that flow rates can be controlled by controlling the accumulations in a distributed manner. We study a *set* of closed-loop congestion control schemes that are based upon the idea of keeping a target accumulation for each flow individually.

[1]This work was supported in part by NSF under contracts ANI-9806660 and ANI-9819112 and a grant from Intel Corp.

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 [14] and Kelly's nonlinear optimization model [8] to demonstrate the equilibrium fairness characteristics of the ACC 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 [4] and other algorithms [14] can be used. Detailed proofs of the ACC stability and fairness are given in the technical report [15].

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 congestion avoidance mechanism attempts to estimate accumulation, and fits into the ACC family. But Vegas often fails to 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 problem 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 in Section III-C. In Section IV we use ns-2 simulations to show the static and dynamic performance of the Monaco scheme. We conclude this paper in Section V by discussing a key concern regarding the ACC model in general, i.e., the scalability of buffer requirement.

II. ACC FLUID MODEL

We define accumulation using a bit-by-bit fluid model and use accumulation to measure and control network congestion. We show that keeping constant accumulation inside network routers for each flow is equivalent to a nonlinear optimization which allocates network capacity proportionally fairly. We then develop a globally stable general control algorithm.

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, flow i's input

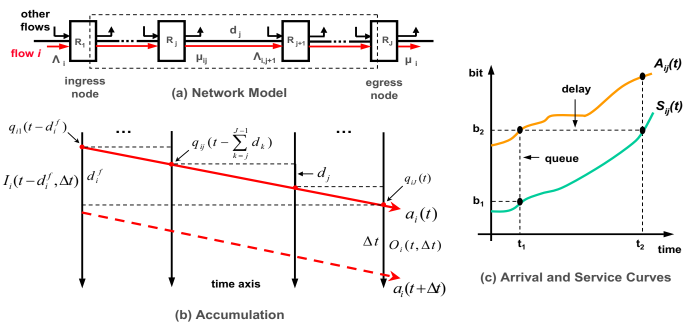


Fig. 1. Network Fluid Model of Accumulation

rate is $\lambda_{ij}(t)$, output rate is $\mu_{ij}(t)$. The propagation delay from node R_j to node R_{j+1} is a constant value d_j.

We define the arrival curve $A_{ij}(t)$ of a flow i at a node R_j as the number of bits of 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, 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[3] or duplication, 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 R_j:

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

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 with fixed delay, 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,

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

where d_j is the propagation delay from R_j to R_{j+1}.

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 the equation includes only those bits backlogged inside the buffers of all nodes on the path, not those stored on transmission links. This definition provides a reference to implement 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 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.

For one FIFO node, it's straight-forward to control flow rates by controlling the number of queued packets [5], since buffered packets decide completely the 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 inside network for each flow*. Note Equations (2) and (6) have a significant difference of the one-way propagation delay d_i^f, which is a constant as long as flow i's route is fixed.

B. *Queuing and Optimization Analysis*

To give a 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 of the ACC model based no [8] [14]. It turns out that ACC 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. 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., $L_i = \{l \in L \mid i$ traverses $l\}$. A link l is shared by a subset I_l of flows where $I_l = \{i \in I \mid i$ traverses $l\}$.

Let's firstly consider from queuing perspective [14]. After the system approaches a steady state (so we can neglect the time variable t in all the previous equations), at any link l the

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

[3]However, an ACC scheme should be robust and respond to unexpected packet losses. See more details in Section III-B.2.

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 $x_i \cdot rtt_{p\,i}$, where $rtt_{p\,i}$ is flow i's round trip propagation delay. Since $w_i = x_i \cdot rtt_i$ where rtt_i is the round trip time observed by flow i, we summarize to get:

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 [8] [11] [9], where 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 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

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

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

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

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 congestion control as shown in Section II-D. Besides, the Lagrange multiplier p_l is a measure of congestion, or price explored in [11], 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. More accurately, we try to maintain a constant target accumulation a_i^* for each flow i by applying a general ACC control algorithm:

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

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 a strictly increasing, differentiable function with a unique root 0 (i.e., only $f(0) = 0$) and $\eta > 0$.

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, where we also show that the above algorithm is globally stable.

By choosing different f functions, we can instantiate the above general algorithm into a set of control policies including the well-known additive-increase-additive-decrease (AIAD) policy popularized by TCP Vegas, an algorithm proposed by Mo and Walrand [14], and a proportional control policy used by the Monaco scheme in the next section, among others.

D. Properties

For any congestion control algorithm, major theoretic concerns are its stability, fairness and queue bound. Stability is to guarantee equilibrium operation of the algorithm. Fairness, either max-min or proportional [8], 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 the following result in [15].

Proposition 3: The accumulation-based control algorithm given by Equation (9) is globally asymptotically 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 sharing that bottleneck. In practice, we need to provide enough buffers to avoid packet loss and make the congestion control protocol robust to such loss, if unavoidably any (see Section III-B). Another way to alleviate this problem is to control aggregate flow in a network edge-to-edge manner, instead of end-to-end for each micro-flow of source-destination pair, since the ACC model can be mapped onto end-to-end hosts or network edge-to-edge (though 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 [10]. We have implemented this option. Due to space limit, we do not include it here. The reader is referred to [15] for details.

Interestingly, as Proposition 3 states (and validated by our experiments), 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 of different dynamic characteristics. In this sense, we regard that the ACC model manifests congestion 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 tries to estimate accumulation and fits into the ACC model. Unfortunately Vegas often fails to provide an unbiased accumulation estimation. Therefore 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 user 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:

- #1: **Stability:** The scheme should converge to an equilibrium in a reasonably dynamic environment with changing demands or capacity;
- #2: **Proportional Fairness:** Given enough buffers, the scheme must achieve proportional fairness and operate without packet loss at the steady state;
- #3: **High Utilization:** When a path is presented with sufficient demand, the scheme should converge around full utilization of the path's resources;
- #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 was proposed as an alternative TCP implementation. It includes several modifications over TCP Reno [7]. 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 [3] [13]). Vegas then estimates the backlog as

$$\hat{a}_V = \left(\frac{cwnd}{rtt_p} - \frac{cwnd}{rtt}\right) \times rtt_p \quad (10)$$

$$= \frac{cwnd}{rtt} \times rtt_q \quad (11)$$

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 estimation for accumulation.

Vegas estimates rtt_p as the minimum RTT measured so far. If the 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 an AIAD policy:

$$cwnd(n+1) = \begin{cases} cwnd(n) + 1 & if \quad \hat{a}_V < \varepsilon_1 \\ cwnd(n) - 1 & if \quad \hat{a}_V > \varepsilon_2 \end{cases} \quad (12)$$

where ε_1 and ε_2 are set to 1 and 3 packets, respectively.

Vegas has several well-known problems: i) *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 [12], violating the goals #1 and #4. Mo et al. [13] 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 III-C; ii) *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 the goal #2; iii) *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 the 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.

1) Monaco: Congestion Estimation Protocol: Let's look at the definition of accumulation in Equation (4). It is the sum of the queued bits of a flow at a sequence of FIFO

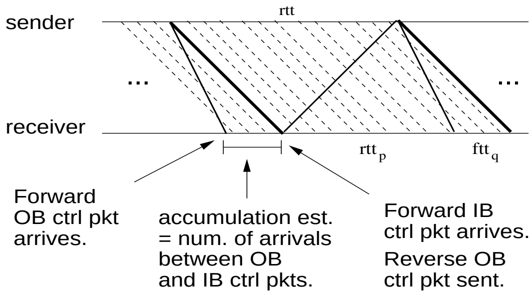


Fig. 2. Monaco Accumulation Estimator

routers, including both ingress and egress nodes[4] as well as intermediate routers. We aim to eliminate the computation at intermediate routers. Actually it is impossible for all nodes R_j $(1 \leq j \leq J)$ to compute *synchronously* their queues $q_{ij}(t - \sum_{k=j}^{J-1} d_k)$ at different times 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 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 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.

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 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.

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

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

We provide below a proportional control policy among a set of what Monaco can use:

$$cwnd(n+1) = cwnd(n) - \kappa \cdot (\hat{a}_M - a^*) \tag{13}$$

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 of Vegas and Monaco

Vegas and Monaco both 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 has severe problems in achieving this objective; Monaco solves known estimation problems.

Vegas estimator operates at *sender* side. According to Equation (10) it actually calculates:

$$\begin{aligned} \hat{a}_V &= \frac{cwnd}{rtt} \times (rtt - rtt_p) & (14) \\ &= \frac{cwnd}{rtt} \times (t_q^f + t_q^b) & (15) \end{aligned}$$

where t_q^f and t_q^b are forward and reverse direction queuing delays, respectively. The above equations imply that Vegas

[4]The reader may think the ingress node as source(sender) and the egress node as destination(receiver).

may suffer from two problems: 1) By Equation (14), if rtt_p is overestimated, then Vegas underestimates accumulation. This leads to extra steady queue in bottlenecks or even persistent congestion. Simulation results in [15] show that Vegas operates with very low utilization of less than 10% and Vegas-k operates with queue increase until loss occurs when there exists rtt_p estimation error. 2) By Equation (15), 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 the forward direction flows have no control over those on reverse direction.

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 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 *receiver* side and thus excludes the effect of reverse path congestion. By measuring the time interval between the OB and IB control packets, Monaco does not explicitly need to estimate the forward direction propagation delay. (Actually the OB control packet provides implicitly 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 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 the OB control packet also makes Monaco adaptive to re-routing since it is sent every RTT. Simulation results in [15] show that, under the same condition, 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 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 forward direction congestion control.

IV. Simulations

In the last section we have shown that Monaco outperforms Vegas. So we focus on evaluating the dynamic and steady state performance of Monaco. We use ns-2 simulations with data packet size of 1000 bytes and target accumulation set to 3000 bytes. We also implement Monaco in Linux kernel v2.2.18 and validate most of simulation results [15]. In brief, in combination with Section III-C, this section shows that Monaco satisfies all the goals outlined in Section III.

A. Single Bottleneck with Dynamic Demands

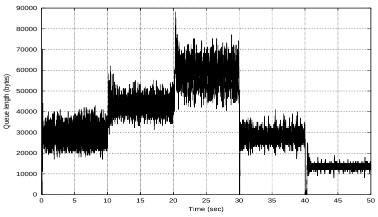
Firstly we consider a single 30Mbps bottleneck with 2ms propagation delay shared by 3 sets of flows using the Monaco scheme. Set 1 has 10 flows starting at 0s and stopping at 30s; Set 2 has 5 flows starting at 10s and stopping at 40s; Set 3 has 5 flows starting at 20s and stopping at 50s. Each source-destination 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 a droptail bottleneck, the other with underprovisioned buffer.

In the first simulation, the bottleneck router has enough buffer of 90 packets, as shown in Figure 3(a1), where there is no packet loss. We randomly pick one flow from each set and draw its individual throughput in Figure 3(a2). 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 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. Bottleneck queue length is depicted in Figure 3(a1) where incoming flows build up a steady queue and flows leave with queue decrease, on average 3 packets for each flow as specified by target accumulation. During the simulation bottleneck utilization always stays around 100%, except two soon-recovered drops during abrupt demand changes at 30s and 40s. This simulation validates that Monaco demonstrates a stable behavior under a dynamic and heterogeneous environment and keeps steady queues inside bottleneck.

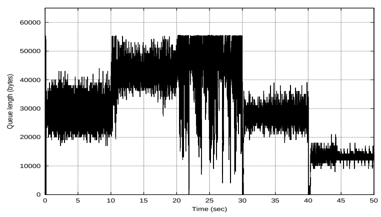
In the second simulation, the droptail bottleneck router buffer is underprovisioned, as illustrated in Figure 3(b1), we can see that the queue length grows to the limit of the whole buffer size of 55 packets, and there is a correspondent packet loss leading to halving of the congestion window during 20s $\sim$ 30s. Consequently, the throughput is more oscillating as seen in Figure 3(b2), 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

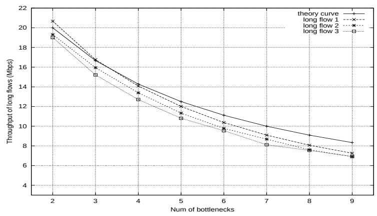
Now we show the steady state performance of Monaco when flow traverses more than one bottleneck. We use a linear topology with multiple congested links. We did a set of simulation experiments by changing the number of bottlenecks N from 2 to 9. There are 3 "long" flows passing all the bottlenecks and 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 simulated for 50s under only one condition with enough buffer provided for all the droptail routers. As already shown in the last subsection, if droptail router buffer is not enough, the Monaco scheme performance degrades.



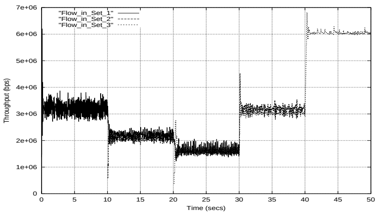
(a1) Bottleneck Queue Length



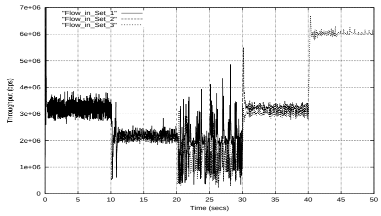
(b1) Bottleneck Queue Length



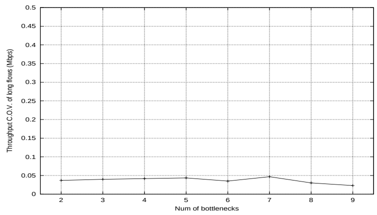
(c1) Long Flow Throughput



(a2) Flow Throughput



(b2) Flow Throughput



(c2) Long Flow Throughput Spread

Fig. 3. Monaco under a Single Bottleneck with Enough (90 packets, a) or Underprovisioned (55 packets, b) Buffer and Multiple Bottlenecks (c)

As illustrated in Figure 3(c1), the steady state throughput curves of all long flows are located near the theoretic one of $100/(3+N)$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 3(c2), which is between 2% and 5% for all cases. This simulation shows that, with enough buffer provisioned, Monaco achieves a proportionally fair bandwidth allocation in a multiple bottleneck case, validating our theoretic results of Proposition 3.

V. Summary

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 – which is its steady state feature. We propose a set of globally stable control algorithms that drive the network toward the 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 scheme Monaco which, with two priority FIFO queues provided by network routers, solves the well-known problems of Vegas. We use a set of simulations to 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 bottleneck routers. With underprovisioned buffer, Monaco's performance is degraded. This buffer scalability problem can be solved by employing the AVQ algorithm running inside the bottleneck, as implemented in [15].

One may ask that if the two-queue support from all bottlenecks, even its complexity is very low, is unrealistic. But 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 ACC scheme performance and its requirement.

By keeping different accumulation for different flows, it's possible to provide differentiated services. These issues are explored in our related research [6].

References

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

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

[3] 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.

[4] 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.

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

[6] D. Harrison, Y. Xia, S. Kalyanaraman and A. Venkatesan. A Closed-loop Scheme for Expected Minimum Rate and Weighted Rate Services. Http://www.rpi.edu/~xiay/pub/acc_qos.ps.gz, *Submitted*, 2002.

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

[8] 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.

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

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

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

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

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

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

[15] Y. Xia, D. Harrison, S. Kalyanaraman, K. Ramachandran and A. Venkatesan. Accumulation-based Congestion Control. *RPI ECSE Tech Report*, Http://www.rpi.edu/~xiay/pub/acc.ps.gz, May 2002.