An Edge-based Framework for Flow Control *

David Harrison†, Yong Xia, Shivkumar Kalyanaraman, Kishore Ramachandran
ECSE and CS† Departments
Rensselaer Polytechnic Institute
{harrisod, xiay, shivkuma, rkishore}@rpi.edu

ABSTRACT

This paper investigates the properties of an edge-based flow control framework that could make it a viable data-plane building block for quality of service (QoS) architectures. We consider two broad categories of flow control: end host-based (e.g., TCP), and network-based (e.g., Fair Queuing, CSFQ) that differ in the choice of nodes that cooperate and where functionality is placed. We propose an edge-based closed-loop (EC) flow control framework in the network-based category in which only network nodes are expected to cooperate. The novelty of the EC-framework lies in its *entire* placement of flow control functionality at edge nodes, except interior core routers provide one or two isolated FIFO queues for the overall framework. The framework is divided into logical components such as congestion estimation and congestion response. Component instances can be combined to form schemes that do not depend upon, but accommodate, packet dropping or marking for congestion detection at core routers. We show that the Vegas is one possible scheme in the EC-framework, and propose two new schemes, "Monaco" and "Riviera". A fluid model analysis is used to develop key concepts, to provide a reference for packet system implementations, and to demonstrate stability, fairness and bounded queue behavior. Simulations illustrate scheme performance, robustness comparisons, potential solutions to pitfalls of existing mechanisms. Architectural mapping of the EC-framework to DiffServ and commercial scenarios like cross-ISP data VPN is discussed.

1. INTRODUCTION

Closed-loop flow control is a well-established end-to-end mechanism for stable, efficient and fair operation of the Internet. In this paper we consider the application of closed-loop flow control to quality of service (QoS) provision. Until recently [10, 25], QoS data plane building blocks have all been open-loop, such as flow shapers, packet schedulers, and buffer management modules.

End-to-end flow control techniques, such as TCP [15] and Congestion Manager [2], assume that end systems cooperate and place *all* flow control functionality at end systems. In this model, consistent with the end-to-end principle [31], network routers may provide optional functionality like active queue management (AQM) for performance enhancement [6, 13]. The primary goal of this model is to *avoid congestion collapse*, i.e., the problem of end-to-end throughput degradation under heavy loads. As usually interpreted, this model does not aim to achieve service isolation or differentiation between participating flows, and cannot function effectively if end systems do not cooperate.

In contrast, network-based flow control models assume that only network nodes cooperate and therefore place all flow control functionality at network nodes. Examples include IntServ [7], DiffServ [5] and *Core*-Stateless Fair Queueing (CSFQ) [33]. The primary goal of the network-based model is *not* the avoidance of congestion collapse, but to provide service isolation or differentiation (a.k.a. QoS functions) at some chosen flow granularity.

Within the network-based model, frameworks may create new system levels (e.g., "edge-routers" and "core-routers") and make choices regarding placement of functionality among these levels. The DiffServ and CSFQ architectures place more per-flow functionality at the edge routers of the network compared to the core routers. This trend is in line with the end-to-end principle [31] because functionality moves to the "highest" system level within the set of cooperating network edge routers, where it can be correctly implemented satisfying the performance constraints.

1.1 Edge-based Flow Control

In this paper, we propose an edge-based closed-loop (EC) framework in the network-based flow control category, to provide a data-plane building block for service differentiation. The architectural novelty of the EC framework lies in its *entire* placement of network-based flow control functionality at edge routers, except as necessary to isolate the EC-traffic, and its use of *closed-loop* schemes in this context.

The elimination of service isolation and/or differentiation functions at core routers implies that service isolation functions must be placed at edge routers to continue providing

*This work was supported in part by National Science Foundation under contracts ANI-9806660, ANI-9819112 and a grant from Intel Corp.

network-wide QoS. However, it is well known that this will lead to inefficient network operation, because unused capacity in the network will not be discovered and utilized. Therefore, in addition, we propose the use of edge-to-edge closed-loop control to be able to match available demand and capacity dynamically. This leads to an architecture where core routers merely provision one or two isolated queues for the overall framework, and multiple services are provisioned entirely from the network edges. In particular, the framework expects no packet marking, buffer management or any other special computation from core routers. These placement-of-functionality decisions distinguishes the framework from RED/ECN [13, 28], Packet-Pair [18], CSFQ [33] or ATM-ABR explicit rate schemes [29].

In addition, the EC-framework proposes to use only closed-loop schemes that do not depend upon packet loss for congestion detection[1]. Loss-based flow control is not a satisfactory building block for service differentiation because it severely limits the dynamic range of service capabilities possible [10, 25]. Packet-loss rate diminishes any bandwidth and loss-rate guarantees possible, interacts with end-to-end transport mechanisms like timeouts and retransmissions; and is very hard to assign carefully unless stateful AQM schemes are available at bottlenecks (which conflicts with our function-placement assumptions).

We define a new measure for network congestion – *accumulation* – that is the time-shifted, distributed sum of the queue contributions of a flow at a sequence of FIFO routers (see section 3). We prove that the behavior of the flow's accumulation measure at a sequence of FIFO routers is very similar to the flow's queuing behavior at a single FIFO router. Since it is well known that service differentiation at a single FIFO router can be effected by controlling the number of buffered packets [14], one can control flow rates in a distributed manner by controlling their accumulations. This aspect is what motivates us to study closed-loop schemes that are based upon accumulation estimation.

We demonstrate that the TCP Vegas [8] congestion avoidance scheme, though proposed originally in an end-to-end context, attempts to estimate accumulation, and fits into our edge-based framework. More generally, in Section 2 we decompose our framework into logical components, instances of which can be combined together to form "schemes". In Section 4.4 we explore the options available at each logical component (e.g., semantics of different congestion estimators and alternative estimation techniques) and the implications of various combinations. In Section 4 we develop two new schemes called "Monaco" and "Riviera" that have different tradeoffs and mechanisms compared to Vegas. Proofs of stability, fairness and queue bound of a scheme are given in Appendix 9.1. In particular, this paper provides resolution to a number of concerns regarding the estimation issues, dynamic stability and robustness of Vegas in Sections 4 and 5, clearing the way to use this class of schemes as a building block for future service architectures.

In summary, this is primarily a paper on the properties of closed-loop flow control, but from the perspective of its potential use as a data-plane QoS building block. Architectural mapping of the framework is discussed in Section 6. A service differentiation example is given in Appendix 9.2.

[1]The schemes will be stable, robust and respond to unexpected packet losses, however.

2. FRAMEWORK

In the EC-framework, routers inside a network are classified into edge and core routers. As illustrated in Figure 1(a), a *unidirectional* traffic flow enters into network at an ingress edge router, traverses a number of core routers, and then leaves network from an egress edge router.

The EC-framework has three components: congestion measure, congestion estimation protocol, and congestion response algorithm. Different flow control schemes make choices in each of these components and put together the entire scheme.

A congestion measure defines the *semantics* of congestion and uses a stream of congestion indications to signal the timing and magnitude of congestion. In our framework, we focus on a congestion measure called accumulation described in the next section. We implicitly detect congestion at the edges, and use in one scheme an explicit rate feedback between edge routers.

The congestion estimation protocol provides an *implementation* of the congestion measure. The implementation may involve ideas like rate/window as a source-throttle, ACK-clocking vs control-packets for feedback, and in-band or out-of-band transmission of control information. Also, we assume that the overall EC-framework is isolated from flows that are not controlled by our algorithms (i.e., one or two isolated queues set aside for the entire framework). This is because the congestion measure used by end-to-end TCP is different from that proposed in the EC-framework. End-to-end flows are either isolated from, or mapped to the flow control provided by the EC-framework.

The congestion response mechanism defines an increase/decrease strategy for the source throttle. A range of traditional algorithms including additive-increase/multiplicative-decrease [9] and additive-increase/additive-decrease [8] algorithms can be used. Alternatively, these algorithms can be applied to a function of both the measured input and rate feedback if any.

We first develop an fluid model analysis for key components in the framework. Then we will describe three schemes (Vegas, Monaco, Riviera), and discuss the properties of key component options to get an improved guidance on how schemes can be instantiated to satisfy desired tradeoffs.

3. FLUID MODEL

In this section we use a bit-by-bit fluid model [11, 27] to illustrate the concepts, algorithms and features for the EC-framework.

3.1 Accumulation

Consider an ordered sequence of FIFO routers $\{R_1, \ldots, R_j, R_{j+1}, \ldots, R_J\}$ along the path of a flow i in Figure 1(a). The flow comes into the ingress edge router R_1 and, after passing some intermediate core routers $R_2, \ldots, R_{J-1}$, goes out from the egress edge router R_J. At time t in any router

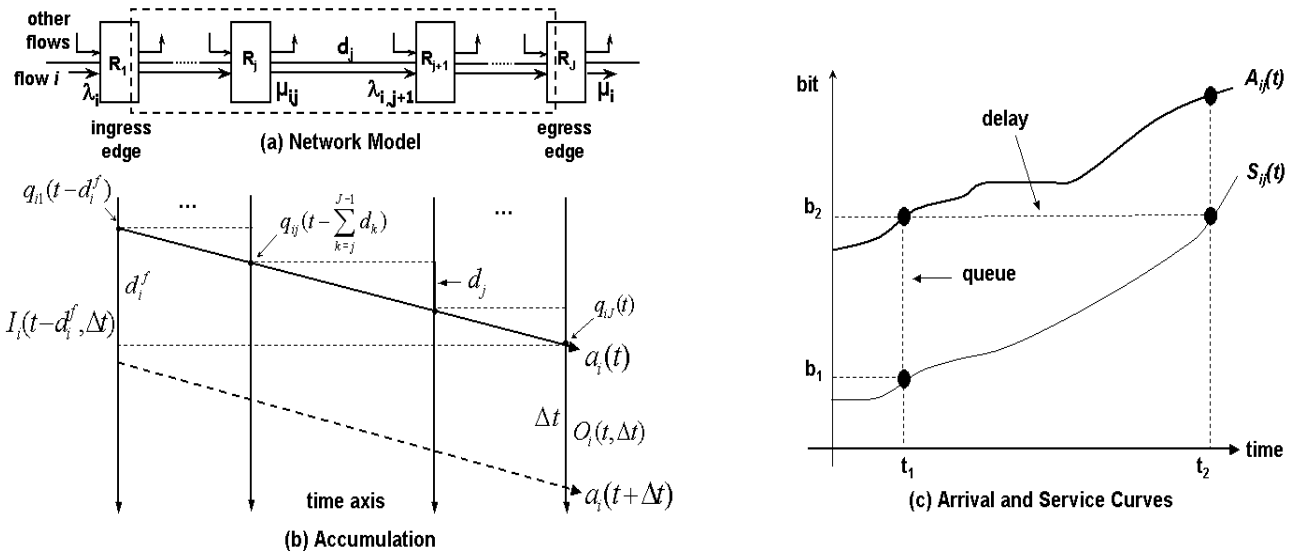


Figure 1: Network Fluid Model

R_j $(1 \le j \le J)$, flow i's input rate is $\lambda_{ij}(t)$, output rate $\mu_{ij}(t)$. The propagation delay from router R_j to R_{j+1} is d_j.

We define the arrival curve $A_{ij}(t)$ of a flow i at a router R_j as the number of bits from that flow have cumulatively arrived at the router up to time t, and similarly the service curve $S_{ij}(t)$ as flow i's bits cumulatively serviced at R_j [11], shown in Figure 1(c). For any FIFO router R_j, both $A_{ij}(t)$ and $S_{ij}(t)$ are continuous [2] and non-decreasing functions. If we assume no packet loss, then at any time t, by definition, flow i's buffered bits $q_{ij}(t)$ in R_j is the difference between $A_{ij}(t)$ and $S_{ij}(t)$:

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

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

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

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

Now consider the flow's queuing behavior at a sequence of FIFO routers. 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 router R_{j+1} is a delayed version of its output rate $\mu_{ij}(t)$ at the upstream neighbor router 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 routers along its path from the ingress router R_1 to the egress router R_J, i.e.,

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

shown as the solid slant line in Figure 1(b). Note this definition includes only those bits backlogged inside routers, not those stored on transmission links. We define flow i's ingress and egress rates as those at the edge routers, respectively:

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

Using (2-5), we calculate the flow's accumulation change

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

where $d_i^f = \sum_{j=1}^{J-1} d_j$ is the forward direction propagation delay of flow i from R_1 all the way down to R_J. Similar as 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 time intervals both of length Δt; while $\overline{\lambda}_i(t - d_i^f, \Delta t)$ and $\overline{\mu}_i(t,\Delta t)$ are correspondent average ingress and egress rates. This result, illustrate in Figure 1(b), shows the change of a flow's accumulation on its path is only related to its input and output at two edge routers.

For one FIFO router, it's straight-forward to control flow rates by controlling the number of queued packets [14]. Comparing equations (2) and (6) we can easily see that a flow's queuing behavior at a sequence of FIFO routers looks similar to that at a single FIFO router. So we can apply the above idea in multiple routers condition to control flow rates by controlling their accumulations.

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

3.2 Control Algorithms

In the EC-framework we use flow accumulation to measure network congestion as well as to probe variation of available bandwidth. If accumulation is low, we increase ingress rate; otherwise, we decrease it to drain accumulation. Specifically, we try to keep a constant ϵ_i of accumulation for every flow i by additively increasing and additively decreasing (AIAD) its ingress rate similar to TCP Vegas [8]:

$$\lambda_i(k+1) = \begin{cases} \lambda_i(k) + \gamma_i & \text{if } a_i(k) < \epsilon_i, \\ \lambda_i(k) - \gamma_i & \text{if } a_i(k) \geq \epsilon_i. \end{cases} \quad (7)$$

where $0 < \gamma_i,\ 0 < \epsilon_i$, and k is the number of a control period which is, ideally, a round trip propagation delay. If γ_i and ϵ_i are equal for all flows, this leads to traditional flow control. Otherwise we can provide different flows differentiated services. We will incorporate AIAD into the Monaco scheme.

Another option is to explicitly use egress rate:

$$\lambda_i(k+1) = \begin{cases} \lambda_i(k) + \alpha_i & \text{if } a_i(k) < \epsilon_i, \\ \mu_i(k) \times \beta_i & \text{if } a_i(k) \geq \epsilon_i. \end{cases} \quad (8)$$

where $0 < \alpha_i,\ 0 \ll \beta_i < 1,\ 0 < \epsilon_i$. We call this algorithm additive increase and multiplicative decrease [9] with egress rate (AIMD-ER) and use it in the Riviera scheme.

3.3 Properties

For any flow control algorithm, major theoretical concerns are its stability, fairness and queue bound. Stability is to guarantee equilibrium operation of the algorithm. Fairness determines allocation of network bandwidth among competing flows, e.g., max-min [27] and proportional fairness [17, 23]. Queue bound provides an upper limit on the router buffer requirement, which is critical for real deployment. Based on the proof in Appendix, we have:

Proposition 1: The flow control algorithm given by equation (8) is stable, weighted proportionally fair, and with bounded queue for any flow in any core router.

Similar result also holds for the algorithm (7), following the line of theory in [22].

4. SCHEMES

We put together component instances to build the edge-based flow control schemes in our framework. The scheme design is guided by the following goals in order of decreasing importance:

Goal 1: **Stability and 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.

Goal 2: **Avoidance of Starvation and Gross Unfairness:** Misbehaving traffic or scheme estimation errors should not lead to starvation or gross unfairness.

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: **Loss Minimization, Proportional Fairness, Delay Minimization:** In the steady state operation, the scheme must operate without loss, with low queueing delay and achieve proportional fairness [17]. In general, given reasonable buffers, the scheme must attempt to minimize instances of packet loss.

Now we describe example flow control schemes in the framework: Vegas, Monaco and Riviera.

4.1 Vegas

The Vegas [8] congestion avoidance scheme was originally proposed in an end-to-end context as an alternative TCP implementation. However, we focus only on its congestion avoidance scheme, which fits well into our framework as a example scheme instance and we refer to it as EC-Vegas.

The Vegas-estimator for "accumulation" was called "backlog" in the original paper, a term we use interchangably in our discussion. For each flow i, the Vegas-estimator takes as input an estimate of i's round trip propagation delay, hereafter called the *basertt*. Vegas then estimates the ith control-loop's backlog as

$$\begin{aligned} \hat{q}_V &= (expected\ rate - actual\ rate) \times basertt \\ &= \left(\frac{cwnd}{basertt} - \frac{cwnd}{rtt}\right) \times basertt. \end{aligned} \quad (9)$$

Vegas estimates the *basertt* 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 *basertt*. The Vegas-estimator is used to adjust its congestion window size, *cwnd*, so that $\hat{q}_V$ approaches a target range $(\varepsilon_1, \varepsilon_2)$. More accurately stated, the sender adjusts the window using a variant version of the AIAD algorithm (7), i.e.,

$$cwnd(k+1) = \begin{cases} cwnd(k) + 1 & if \quad \hat{q}_V < \varepsilon_1 \\ cwnd(k) - 1 & if \quad \hat{q}_V > \varepsilon_2 \end{cases} \quad (10)$$

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

TCP Vegas is an ACK-clocked window-based scheme. But, the Vegas-estimator as specified above does not require an ack-stream. However, one of the Vegas implementations uses the average RTT experienced by all ACKs in the previous window to compute the backlog. To implement RTT averaging at this timescale requires an ACK stream.

Thus, for EC-Vegas, we need to introduce either a new edge-to-edge ACK flow, or a control packet to sample RTT. Edge-to-edge ACK flows have also been proposed by Wang and Kung [19], but they propose to implement TCP's loss-based congestion control for the edge-to-edge traffic. Such ACK-flows represent more overhead in general compared to the control-packets used in Riviera and Monaco. Vegas has several known problems:

Basertt **Estimation Errors:** Suppose re-routing of a flow increases its *basertt*. Vegas misinterprets an increase in *basertt* as congestion and backs off. This can result in gross unfairness which is a violation of Goal

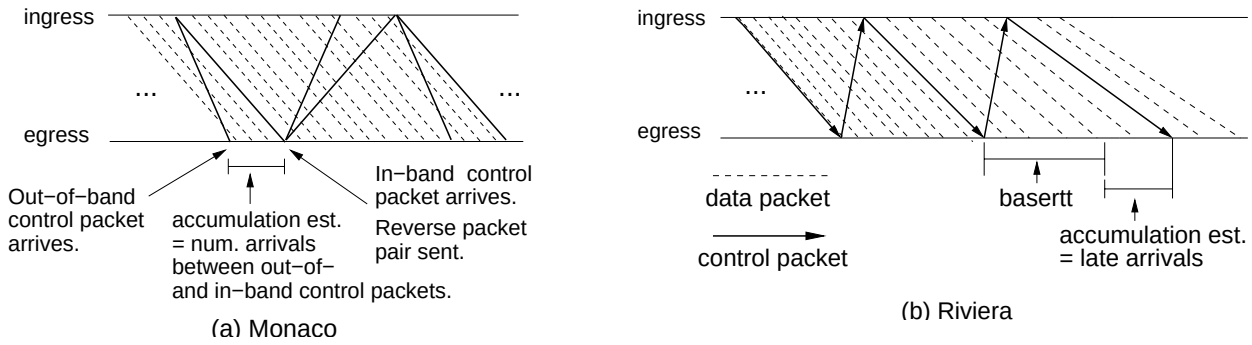


Figure 2: Accumulation Estimators

2 and makes Vegas as originally designed unsuitable for the EC framework. Mo and Walrand [24] suggest limiting the history on the *basertt* estimate by using the minimum of the last k RTT samples. We refer to this variant as the "Vegas-k" scheme, which avoids the gross unfairness issue. But we show in Section 4.4 that since Vegas uses an AIAD policy, it cannot guarantee queue drain at intermediate bottlenecks within k RTTs. Hence, this policy can lead to unbounded queue which introduces persistent congestion [22], violating Goal 1.

Basertt **with Standing Queues:** When a flow arrives at a bottleneck with a standing queue, it obtains an exaggerated *basertt* estimate. The flow then adjusts its window size to incur an extra backlog between ε_1 and ε_2 in addition to the standing queue, violating Goal 4.

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.

4.2 Monaco

Monaco is an EC-framework scheme that emulates the accumulation concept described in Section 3.1, while being robust to data/control-information losses and avoiding issues such *basertt* sensitivities and reverse path congestion.

4.2.1 Monaco: Congestion Estimation Protocol

To estimate accumulation, Monaco generates a pair of back-to-back control packets once per RTT at the ingress router as shown in Figure 2(a). One control packet is sent in-band (IB) and the other out-of-band (OB). The OB control packets skip queues in the network by passing through a separate dedicated high priority queue. Assuming the OB queues to be minimal as only other OB control packets share it, such packets experience only the forward propagation delay d_i^f. The IB control packet goes along with regular data packets and reaches the destination after experiencing the current queueing delay in the network. The Monaco-estimator counts the number of bytes arriving between the IB and OB control packets, see Figure 2(a). Observe that in a fluid model, this is an exact measure of the true accumulation. In particular, note that in Figure 2(a), the number of dashed lines cut by the OB control packet is the notion of accumulation defined by equation (4). This is exactly equal to the number of arrivals at the receiver after the OB control packet, but before the IB control packet.

The IB control packet carries a byte count and control-packet sequence number. If the egress receives fewer bytes than were transmitted, then a packet loss is detected. The OB control packet carries the same control-packet sequence number as the associated IB control packet and one additional piece of information: congestion feedback, i.e., flags denoting whether the flow throttle should increase, decrease, or decrease due to loss. Monaco also sends congestion feedback on the OB control packet. Observe, however that the subsequent pair of control packets is generated only after the arrival of the IB control packet at the ingress edge.

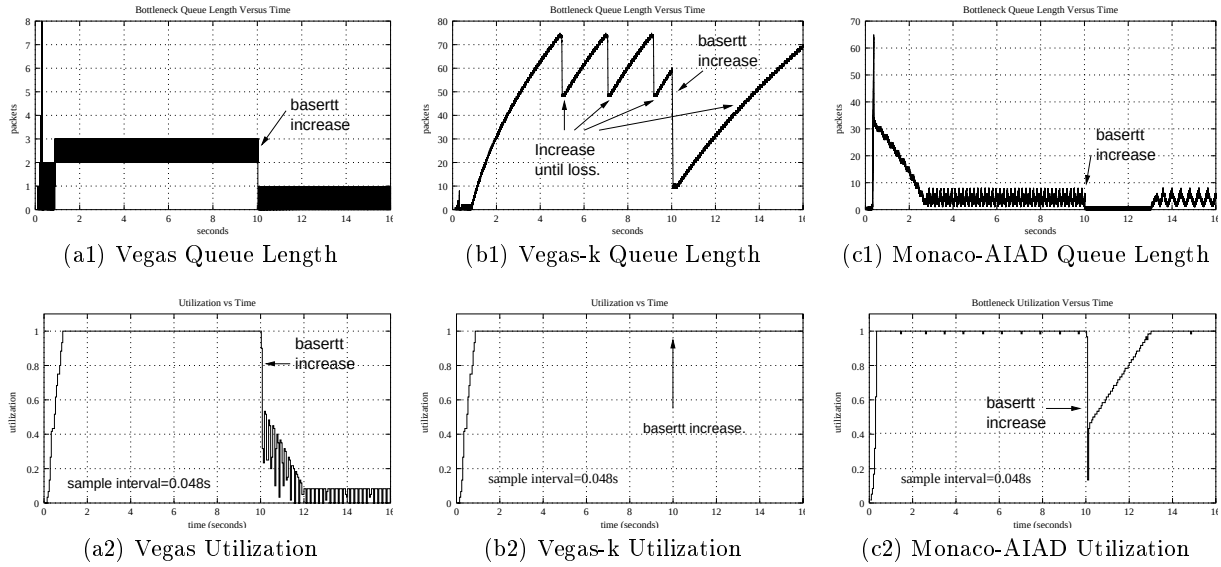
If either control-packet itself is lost, then the source times out and sends a new pair of control-packets with a larger sequence number. The timer for control-packet retransmission is set similar to that of TCP. These routine reliability enhancements are similar to those in the Congestion Manager protocol [2]. Also note that Monaco is designed to avoid the usage of clock values of the sender and receiver in any single computation (the estimator is just a simple count). This design avoids any issues with clock resolution, skew or drift between ingress and egress.

Monaco mechanisms also remove the need for *basertt* measurement and associated problems observed in Vegas. However, the Monaco-estimator requires an additional queue at potential bottlenecks. We deem that this requirement is not costly given that at least one separate queue is already required to isolate EC traffic from other non-EC traffic traffic classes. In summary, we propose Monaco congestion estimation protocol as an alternative to the Vegas congestion estimation protocol in the EC-framework because it addresses all the outstanding problems with Vegas-estimator, at minimal additional cost.

4.2.2 Monaco: Congestion Response Algorithm

We have several choices for the increase/decrease policy used by Monaco. One option is AIAD window-based policy augmented by rate-modulated pacing. Monaco updates its window value according to algorithm (10), and it clocks packets out using a shaper with a rate value of of $cwnd$/RTT.

Another option is a simplified discrete approximation of one



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

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

Figure 3: *Basertt* **Estimation Error**

policy in Mo and Walrand [23]:

$$cwnd(k+1) = cwnd(k) - \eta \times (\hat{q}_M - target) \qquad (11)$$

where $\hat{q}_M$ is Monaco accumulation estimation, $target$ is a target backlog in the path akin to Vegas' ε_1 and ε_2, η is a parameter in $(0, 1)$. Both algorithms converge on proportional fairness as illustrated in our simulations in which we compare AIAD and Mo and Walrand's policies denoted Monaco-AIAD and Monaco-MW respectively.

4.3 Riviera

To eliminate the requirement of a high priority queue for out-of-band control packets, we designed another scheme called "Riviera" that has similarities to both Monaco and Vegas but has a different set of tradeoffs.

4.3.1 *Riviera: Congestion Estimation Protocol*

Unlike Monaco, Riviera uses only one IB control packet which "bounces" between ingress and egress. Therefore Riviera does not require the extra OB control-packet queue used in Monaco. This control-packet is used for all the key functions: accumulation estimation, *basertt* estimation, feedback generation and data-packet loss detection. The Riviera-estimator for accumulation is illustrated in Figure 2(b). The egress router maintains a timer set to *basertt*, which is the estimate of round-trip propagation delay as in Vegas. The timer is started after the last arrival of the control packet at the receiver. The Riviera-estimate of accumulation is simply the number (count) of packet arrivals after the timer expires till the next control packet arrives assuming the control packet was not lost. We also refer to these as "late arrivals."

Assuming no reverse path congestion, the expiry of the timer is roughly the same point at which Monaco's OB control packet would have arrived. Hence, in this case, following the argument for Monaco, the Riviera-estimator would closely approximate the true accumulation of the flow. Riviera uses the same thresholding method as Monaco to detect congestion.

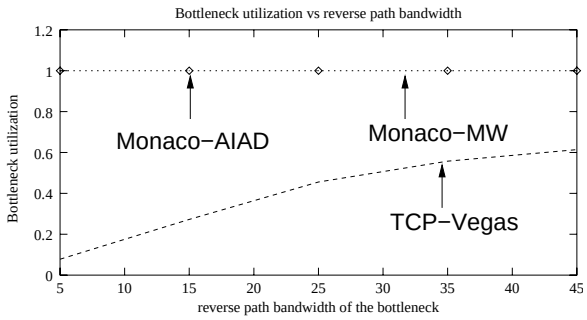


Figure 4: Reverse Path Congestion

The loss-detection mechanisms for control-packets and data-packets and retransmission mechanisms for control-packets are virtually identical to Monaco, except for the fact that a single IB control packet is used in Riviera instead of a pair of IB and OB control packets in Monaco. As mentioned earlier, these simple reliability mechanisms are also found in earlier schemes [2]). Observe that even though a timer is used in the receiver, the RTT estimates are obtained by using only the local clocks at the ingress and egress based upon successive arrivals of the control packet. Therefore, like Monaco, Riviera avoids issues related to clock skew and clock drift.

4.3.2 *Riviera: Congestion Response Algorithm*

Riviera recognizes the interactions between the AIAD policy chosen by Vegas and the *basertt* estimation issues. There-

fore it chooses a more conservative AIMD-ER policy:

$$\lambda_i(k+1) = \begin{cases} \min\{\lambda_i(k),\ \mu_i(k)\} + \alpha_i & if\ \hat{q}_R < \varepsilon_1 \\ \min\{\lambda_i(k),\ \beta_i \cdot \mu_i(k)\} & if\ \hat{q}_R > \varepsilon_2 \end{cases} \tag{12}$$

where $\hat{q}_R$ is Riviera-estimation and other parameters similar as in algorithms (8) and (10). This policy, especially with β_i provisioned conservatively, allows a higher probability of queue drain within k RTT samples (which is the period Vegas-k used to estimate *basertt*). The AIMD-ER algorithm also approximates ack-clocking since it increases and decreases relative to the egress rate. We refer to this feature as "rate-clocking".

As discussed below, Riviera faces similar problems as Vegas. Thus in our simulations, we do not consider Riviera and focus on the differences between Monaco and Vegas. We provide a theoretical analysis for Riviera in Appendix.

4.4 Comparisons

Vegas, Riviera and Monaco aim to accurately estimate accumulation, assuming different support from core routers. If *basertt* can be obtained precisely and there is no reverse path congestion, then by Little's law, they all give unbiased accumulation estimation on average. But in practice Vegas and Riviera often have severe problems in achieving this objective; Monaco solves known estimation problems.

Vegas estimator operates at ingress side. According to equation (9) it actually calculates:

$$\hat{q}_V = \frac{cwnd}{rtt} \times (rtt - basertt) \tag{13}$$

$$= \frac{cwnd}{rtt} \times (t_q^f + t_q^b) \tag{14}$$

where t_q^f and t_q^b are forward and reverse direction queuing delays, respectively. The above equations show Vegas may suffer from two problems: 1) By equation (13), if *basertt* is overestimated, then Vegas underestimates accumulation. This might lead to steady queue in bottlenecks or even persistent congestion. Results for a single bottleneck topology are shown in Figures 3(a) and (b), where *basertt* estimation error is introduced by a sudden *basertt* change at time 10s. Vegas operates with very low utilization of less than 10% and Vegas-k operates with queue increase until loss occurs. 2) By equation (14), if there exists reverse direction queuing delay, i.e., $t_q^b > 0$, then Vegas overestimates accumulation. This leads to underutilization and is hard to handle because Vegas has no control over reverse direction flows, as shown in Figure 4 where Vegas utilization is only 10% $\sim$ 60%.

Riviera faces the same problems as Vegas. It tries to appropriately set β_i in AIMD-ER algorithm (12) to periodically drain the bottleneck queue according to equation (6), thus increases the possibility of successfully sampling *basertt*. In our experiments we tried several increase/decrease policies. We found it's still hard to get precise samples of *basertt* when there are many flows.

Due to the above problems, both Vegas and Riviera fall short of qualifying as a general building block for service differentiation, because we expect to achieve service differentiation by maintaining differential accumulations of flows inside the system in 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 in k RTTs, and hence dynamic *basertt* estimation would be impossible with in-band control packets. In summary, the sensitivity issues with Vegas and Riviera point to a *fundamental* problem with the in-band techniques for accumulation estimation.

Monaco solves both problems. Monaco estimator operates at egress 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 (actually the out-of-band control packets provide implicitly this value). More specifically, Monaco implements a rate-paced window control algorithm to smooth out incoming traffic. So the time difference between the in- and out-of-band packets gives a sample of the current forward direction queuing delay. By Little's law, the number of data packets arriving during this time period is the backlogged packets along the path. In the fluid model, this is precisely the flow's accumulation. In real system in which packet transmission is not preemptive, we might have a half packet estimation error on average at a bottleneck. Considering this possible measurement error, we set the threshold as a range of 1 to 3 packets, instead of using a single value. This improves the robustness of the scheme. Another advantage of using out-of-band control packet is adaptive to re-routing since it is sent every RTT. As shown in Figure 3(c) and Figure 4, after a brief transient period, Monaco operates at around 100% utilization with very low queue. So it's immune to *basertt* estimation inaccuracy and reverse path congestion.

5. SIMULATIONS

Our objectives in this section are to illustrate:

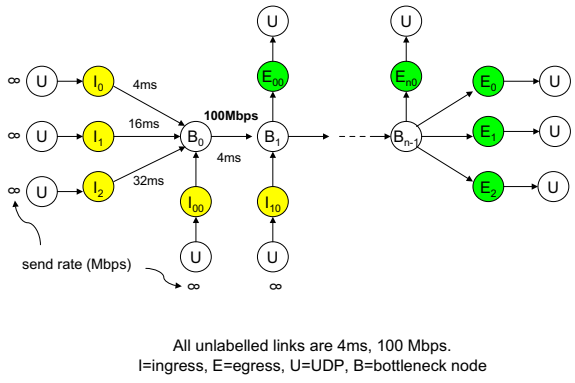
a) Basic EC-scheme steady state performance (utilization, throughput, proportional fairness, throughput variance) in Section 5.1. We use a workload of infinite demand, no interaction with end-to-end transport mechanisms like timeouts etc., and no background non-EC flows. We use a topology having multiple bottlenecks and heterogeneous RTTs.

b) Performance with a complex workload to illustrate safe and robust behavior of the EC-framework in more realistic settings in Section 5.2. The workload includes a mix of web-like short flows and long FTP flows, varies the overall demand, has a large number of total flows at any bottleneck, and has background non-EC flows.

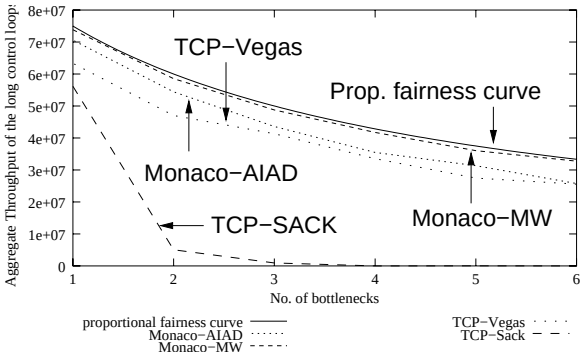
The simulations also show performance of Monaco-AIAD and Monaco-MW compared to Vegas, Vegas-k, and TCP SACK. In brief, this section, in combination with Section 4.4 shows that the EC-framework (and the Monaco scheme) satisfies all the goals outlined in Section 4 and supports a mix of web/FTP traffic efficiently. In all simulations we use `ns-2` [26] with the parameters given in Table 1.

5.1 Steady-state Multiple Bottleneck Case

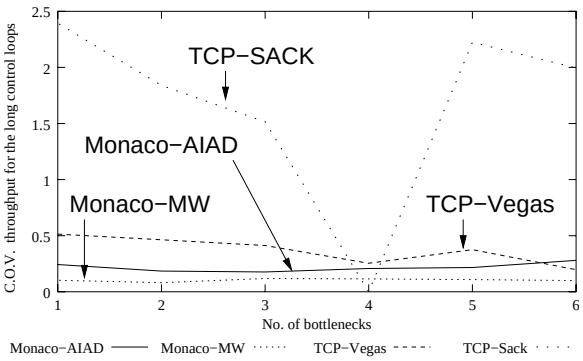
To illustrate steady state scheme behavior, we use the multi-bottleneck network topology in Figure 5(a), with edge nodes between the end systems and the interior nodes. We also have heterogeneous propagation delays along various paths



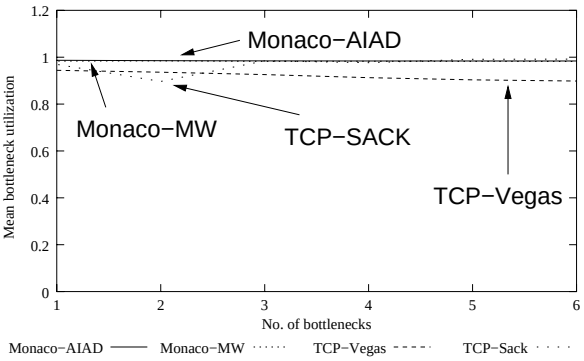
(a) Topology



(b) Aggregate Throughput of the Long Path

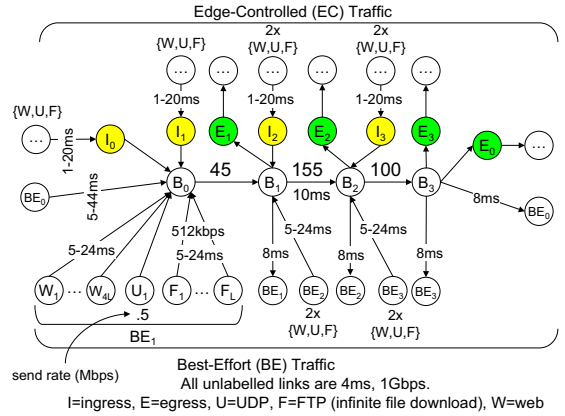


(c) Throughput C.O.V. of Loops on the Long Path

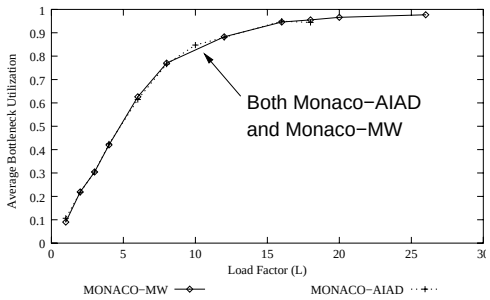


(d) Mean Utilization

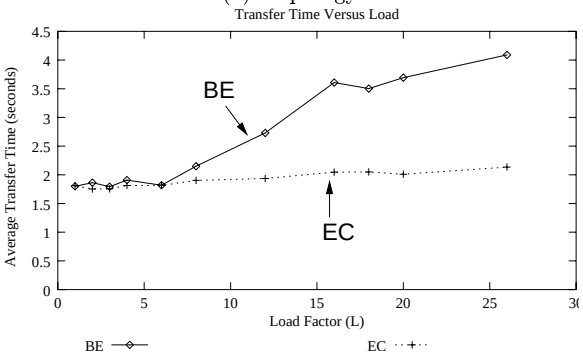
Figure 5: Multiple Bottlenecks



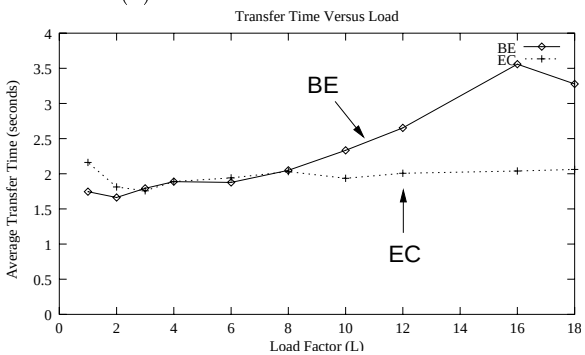
(a) Topology



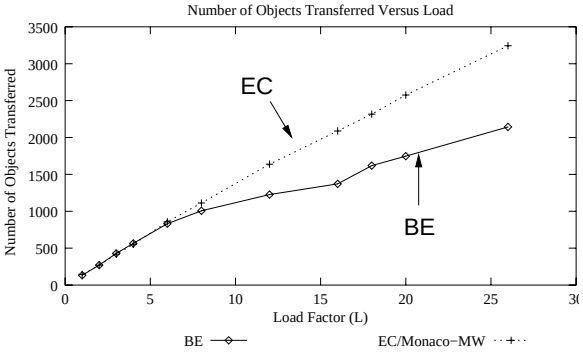
(b) Mean Bottleneck Utilization



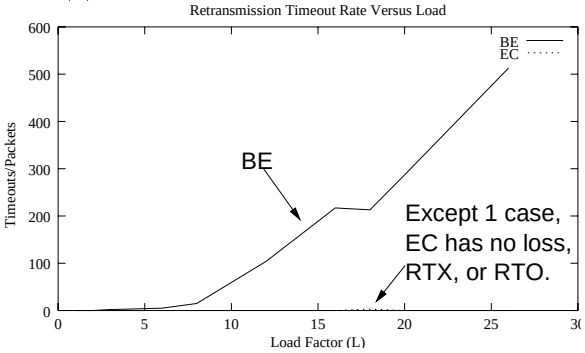
(c) Monaco-MW and BE Transfer Times



(d) Monaco-AIAD and BE Transfer Times

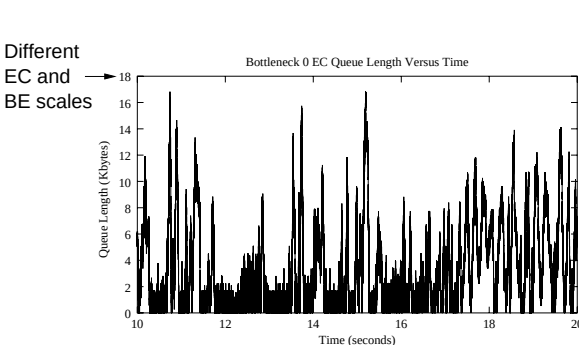


(e) Monaco-MW Number of Objects Transferred

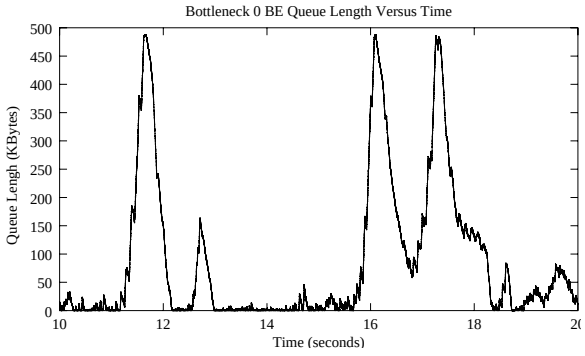


(f) Monaco-MW Web TCP Retransmissions

Figure 6: Web File Transfer Results



(a) Monaco-MW Queue



(b) BE Queue

Figure 7: Transient Queueing for Web Transfers for L=8

Table 1: Simulation Parameters

Parameter	Value
η	0.5
max burst size, σ	1000 bytes
target accumulation	3000 bytes
UDP packet size	1000 bytes
TCP data packet size	576 bytes

to illustrate the fact that our schemes are not sensitive to this issue. We use infinite demand from UDP sources at end systems to avoid interactions with transport-layer mechanisms.

The only difference in the TCP Vegas and TCP SACK simulations is that they do not have edge nodes. For EC-framework schemes, we use large enough buffers to avoid loss. However, because TCP SACK requires loss to detect congestion, we use a buffer of bandwidth-*basertt* product computed from the *basertt* of the $I_0 \rightarrow E_0$ path in Figure 5(a).

Figure 5(b) shows the aggregate throughput of the long paths ($I_0 \rightarrow E_0, I_1 \rightarrow E_1, I_2 \rightarrow E_2$) versus the number of bottlenecks encountered on those paths, benchmarked against the theoretical curve for proportional fairness. It demonstrates that the EC-schemes (Monaco-AIAD, Monaco-MW and Vegas) approach proportional fairness, satisfying Goal 4 in Section 4, whereas TCP SACK does not.

Figure 5(c) shows the Coefficient Of Variation (C.O.V.) in throughput defined as standard deviation divided by the mean, a measure of throughput spread. Figure 5(d) shows the average utilization achieved. Monaco and TCP SACK achieves slightly higher utilization than TCP Vegas. Monaco's throughput C.O.V. is lower than TCP Vegas, which is much lower than TCP SACK.

Monaco outperforms Vegas by a small margin in each case due to its superior accumulation estimator. Section 4.4 illustrated a more decisive advantage of Monaco over Vegas, which is why we recommend Monaco as the default EC-framework scheme. The TCP SACK underperformance in these results may be attributed in part to synchronization effects and timeouts. We present it here to point out that it does not achieve proportional fairness.

5.2 Performance with Web Workload

To demonstrate Monaco's behavior under more complex and realistic conditions, we use a similar topology in Figure 6(a) as the last section, but change the workload to be a mix of end-to-end, web-like short TCP flows (mice), long FTP flows (elephants), and constant-bit rate UDP flows mapped onto the edge-to-edge EC-flows. The UDP traffic occupies a constant fraction of the bottleneck. Each bottleneck sees a non-trivial number total end-to-end flows, either mapped onto the EC-flows, or isolated into a separate queues as non-EC flows. In addition, to illustrate the impact of intelligent edge techniques, we introduce isolation between UDP, web and FTP flows only at the edge before multiplexing them onto an EC-loop. Moreover, a transport-aware technique, TCP Rate Control [16], is introduced to illustrate the performance customization potential possible at edge-nodes. The goal of introducing such complexity is to illustrate the performance by turning on all potential options in the EC-framework and to re-iterate that it is safe and beneficial to map end-to-end flows over the EC-schemes.

We refer to the non-EC flows as best-effort (BE) traffic and we isolate BE from EC traffic by using Deficit Round Robin (DRR) [32] schedulers at each bottleneck. DRR is chosen because it is one of the simplest known fair queueing schedulers. To implement the Monaco backlog estimator, we break the DRR's EC-class into two queues: high priority for the out-of-band control packets, and low priority for in-band and data packets. The two priority queues together receive only one pre-allocated share from the DRR scheduler. Each ingress edge in addition, uses DRR to achieve isolation between web, FTP and UDP end-to-end flows mapped onto an EC-loop.

All three queues multiplexing onto a bottleneck link share 500KBPS, and similarly for the three queues multiplexing onto an EC-loop at the edge. As with standard DRR, when buffers are exhausted, packets are dropped from the largest of the three queues. Unless EC erroneously admits more control packets then in-band plus data packets, bottleneck loss will usually occur in either the EC low priority or BE queues.

For the web and FTP traffic mapped to the EC-loop, each ingress applies the TCP Rate Control (TCPR) technique. TCPR sets the receiver advertised window in passing acknowledgements to bound each connection's window size to a fair share of the sum of the FTP and web queues' average service rates. The separate web queue simply allows mice arriving at a bottleneck dominated by elephants to come up quickly. In other words, with TCPR, it is not necessary to induce packet loss at the edge-nodes to constrain each end-to-end TCP flow. Therefore, the coupling of EC-framework control between the edge nodes, and TCP rate control at the ingress edge allows TCP to experience zero end-to-end congestive loss when there is sufficient buffer at all bottlenecks in the path. Note however that TCPR requires access to TCP ACK flow headers, and can only be applied if such access is available at EC-edge nodes (e.g., site-to-site VPN in Section 6).

To simulate web traffic, we use Barford and Crovella's model [3]. We feed an infinite supply of packets to a TCP SACK connection to simulate FTP. To achieve a roughly even mix of web and FTP packets entering each bottleneck, we use a (empirically determined) ratio of 8 web sources to one Mbps constrained FTP connection. We vary aggregate demand on the interior bottlenecks by scaling the number of FTP and web sources while retaining the same demand ratio. The total UDP traffic remains a constant. To maintain comparability between the EC and BE traffic classes, we mirror the workloads and paths between the EC and BE traffic classes. As shown in Figure 6(a), load scaling factor L denotes the number of FTP flows and $4L$ the number of web flows in each group BE_i.

Figure 6(c) and (d) confirm that Monaco improves transfer times as the load increases with no negative effect at low

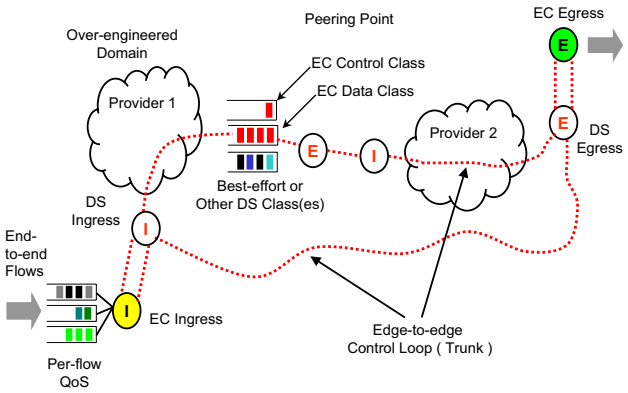


Figure 8: QoS across Multi-ISPs

loads, while Figure 6(e) demonstrates that the improvement in transfer times is not achieved by reducing demand. Here we define a "web object" as the results of the HTTP transfers initiated by a user action (i.e., click, enter URL). It is also unlikely that EC is stealing from the BE class due to the DRR scheduler.

Also worth noting is that neither Monaco-AIAD or Monaco-MW incurs loss in any of these simulations. Though under heavy loads, both Monaco-MW (3 timeouts for L=18) and Monaco-AIAD (33 timeouts for L=18) falsely trigger timeouts in one case. The lack of loss can be attributed to Monaco's tight control on queue length despite substantial variance (see Figure 7(a)).

6. ARCHITECTURE MAPPING

In this section, we discuss how to map the EC-framework onto QoS and overlay architectures. The EC-framework is *complementary* to, and does not necessarily compete with current technologies like DiffServ, MPLS or ATM/Frame-Relay networks. We discuss three examples to illustrate the potential: mapping to DiffServ, and mapping to provide a site-to-site Virtual Private Network (VPN) service that crosses multiple ISP boundaries, and an Overlay QoS service.

The EC-framework naturally maps to the DiffServ (DS) architecture [5]. In such a mapping, the EC-edge nodes can become DS edge nodes, and implement the isolation and closed-loop functions. The DS-byte would carry a code-point that maps to a per-hop behavior (PHB) implementation at interior nodes. The PHB isolates all aggregate EC-framework traffic into one queue (Vegas, Riviera) or two queues (Monaco where the second, higher priority queue is for out-of-band control packets). We are currently investigating mechanisms that allow the EC-framework to realize a variety of bandwidth sharing objectives. The EC-framework can also be used in conjunction with current DS PHBs and services. In particular, the EC-framework can operate the interior DS network in a near lossless manner, and migrate the issues of per-flow packet-loss assignment to the edge nodes where more stateful and application-aware methods can lead to superior per-flow performance customization.

The EC-framework simplifies the bandwidth services provision across multiple provider networks. Figure 8 shows that an EC-loop can originate in one ISP and terminate in another, provide they agree to cooperate. Cooperating ISPs need to negotiate contracts on aggregate traffic characteristics, and do a *one-time* provisioning of the EC-class at potential bottlenecks along the paths taken by EC-traffic. Fine-grained control of packet loss and bandwidth assignment, and any novel services may be provided purely at EC-edges. Monitoring of aggregate traffic and potential punishment of any misbehavior at the ISP boundaries is enough to enforce contractual agreements between ISPs.

Another way to provision QoS over multi-provider boundaries is to introduce an "Overlay QoS" provider who owns several nodes in the network (e.g., Akamai has over 1000 points of presence in 63 countries), and who buys commodity point-to-point bandwidth or Service Level Agreement (SLA) that crosses only a single physical service provider. It is well known that such overlay networks could, with high probability, route traffic not substantially worse than current Internet BGP routing [1, 30]. QoS hungry end-to-end applications could be mapped to such networks at the edges, and their performance managed completely from edges using a lightweight framework like EC. The overlay edge is a point where network QoS management and application intelligence meet (see Section 7). We expect such overlay QoS-based applications to be pervasive in the future.

7. SUMMARY AND FUTURE WORK

In this paper we propose an edge-based flow control (EC) framework. The EC-framework places all flow control functions at network edge routers, keeping core routers' forwarding algorithm simple (one or two FIFO queues for the entire framework). Using closed-loop EC-schemes leads to higher efficiency, and service differentiation capabilities operating at O(RTT) timescales, as discussed in Appendix 9.2. A new congestion measure, accumulation, is developed because loss-based congestion detection severely limits the range of potential QoS capabilities. Although the ultimate objective of this work is to provide QoS, this paper exclusively focuses on the enabling closed-loop flow control properties. In summary, the main contributions of this paper are:

- a modular framework to generate flow control schemes by combining framework components;
- a set of schemes with provable stability, fairness and queue bound;
- a mathematically defined, physically meaningful concept of backlogged packets, accumulation of a flow;
- a protocol realization of the accumulation estimation in packet-switched networks that emulates accumulation calculation in the fluid model;
- the framework positioned as a closed-loop data-plan QoS building block for future services.

The placement of functions at the edge has interesting architectural implications. The number of nodes which need to be upgraded for QoS is smaller (only edge nodes). The removal of functions from interior nodes means that they are not only stateless (like CSFQ), but also do not need to

support new computation and do not need to be configured (or signaled). In other words, interior nodes can truly focus on their core task of packet forwarding (especially when they are resource constrained as in cheap overlay network routers). End systems may be un-cooperative, and may not even support congestion control, but their traffic can be effectively isolated and punished at the edges. Policing or penalty-box functions are not needed anywhere else in the network. The edge is also "closer" to the end systems. Therefore, it becomes more likely that applications can convey their intelligence to the edge and participate in the QoS assignment (or customization) process. This would lead to a class of edge-based low-cost customized QoS services for applications even though the path may have multiple bottlenecks other than the edge. These issues will be explored in a future paper.

8. REFERENCES

[1] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. ACM SOSP'01*, Oct 2001.

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

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

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

[5] S. Blake et al. An Architecture for Differentiated Services. *IETF RFC 2475*, Dec 1998.

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

[7] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. *IETF RFC 1633*, Jun 1994.

[8] 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.

[9] 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.

[10] J. Crowcroft and P. Oechslin. Differentiated End-to-End Internet Services Using a Weighted Proportional Fair Sharing TCP. *ACM Computer Communication Review*, 28(3), Jul 1998.

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

[12] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulations of a Fair Queueing Algorithm. In *Proc. SIGCOMM'89*, Sept 1989.

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

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

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

[16] S. Karandikar et al. TCP Rate Control. *ACM Computer Communication Review*, 30(1), Jan 2000.

[17] 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.

[18] S. Keshav. A Control-theoretic Approach to Flow Control. In *Proc. SIGCOMM'91*, Sept 1991.

[19] H. Kung and S. Wang. TCP Trunking: Design, Implementation and Performance. In *Proc. ICNP'99*, Oct 1999.

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

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

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

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

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

[25] T. Nandagopal et al. Scalable Service Differentiation using Purely End-to-End Mechanisms: Features and Limitations. In *IPQoS'00*, Jun 2000.

[26] Network Similator 2. Http://www.isi.edu/nsnam/ns/.

[27] 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.

[28] K. Ramakrishnan, and S. Floyd. A Proposal to Add Explicit Congestion Notification (ECN) to IP. *IETF RFC 2481*, Jan 1999.

[29] S. Kalyanaraman et al. The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks. *IEEE/ACM Trans. on Networking*, 8(1), Feb 2000.

[30] S. Savage et al. The End-to-end Effects of Internet Path Selection. In *Proc. SIGCOMM'99*, Sept 1999.

[31] J. Saltzer, D. Reed and D. Clark. End-To-End Arguments In System Design. *ACM Trans. on Computer Systems*, 2(4):277-288, Nov 1984.

[32] M. Shreedhar, George Varghese. Efficient Fair Queuing using Deficit Round Robin. In *Proc. SIGCOMM'95*, Aug 1995.

[33] I. Stoica, S. Shenker and H. Zhang. *Core*-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proc. SIGCOMM'98*, Sept 1998.

9. APPENDIX

In this appendix we provide a fluid model analysis for Riviera and a service differentiation example based on Monaco.

9.1 Properties

Following [17, 20, 21], we prove briefly the stability, fairness and queue bound for Riviera using the proposed fluid model. Similar results [22] exist for Monaco but are not provided here. Detailed proof will be provided in a tech-report.

9.1.1 Stability

Consider a network of a set $L = \{1, \ldots, L\}$ of links, shared by a set $I = \{1, \ldots, I\}$ of flows. Link $l \in L$ has capacity c_l. Flow $i \in I$ passes a route r_i consisting a subset of links, *i.e.*, $r_i = \{l \in L \mid i \text{ uses } l\}$.

Suppose the probability for flow i to detect congestion is

$$p_i(k) = Prob[\, a_i(k) \geq \epsilon_i \,], \tag{15}$$

then we have, by algorithm (8),

$$\lambda_i(k+1) = [1 - p_i(k)] \cdot [\lambda_i(k) + \alpha_i] + p_i(k) \cdot \beta_i \mu_i(k).$$

Divided by the length of the control period which is, under ideal feedback condition, the round-trip propagation delay $d_i^r = d_i^f + d_i^b$, where d_i^f and d_i^b are the forward and backward direction propagation delay of flow i, respectively, we get

$$\begin{aligned} \dot{\lambda}_i(k) &\doteq \frac{\lambda_i(k+1) - \lambda_i(k)}{d_i^r} \\ &= \frac{\alpha_i[1 - p_i(k)]}{d_i^r} - \frac{p_i(k)}{d_i^r} \cdot [\lambda_i(k) - \beta_i \mu_i(k)] \end{aligned}$$

or simply

$$\dot{\lambda}_i = m_i \cdot [w_i - s_i(\, \lambda_i - \beta_i \mu_i \,)] \tag{16}$$

where

$$\begin{aligned} m_i &= 1 - p_i(k) \\ w_i &= \alpha_i \,/\, d_i^r \\ s_i &= p_i(k) \,/\, d_i^r [1 - p_i(k)] \end{aligned}$$

are all non-negative. This differential equation shows the dynamics of the AIMD-ER algorithm (8). We prove its stability by constructing a Lyaponov function of system states.

Define a function of all flows' ingress rates [17, 20]

$$U(\overrightarrow{\lambda}) \triangleq \sum_{i \in I} [w_i \log \lambda_i - s_i(1 - \beta_i)\lambda_i] - s_i \beta_i \sum_{l \in L} \int_0^{\lambda^l} p_l(c_l, x)\, dx \tag{17}$$

where the penalty $p_l(c_l, x)$ at link l is a non-negative, continuous, increasing function of x:

$$\begin{cases} p_l(c_l, x) = 0 & \text{if } x \leq c_l, \\ p_l(c_l, x) > 0 & \text{if } x > c_l. \end{cases} \tag{18}$$

It also satisfies $\sum_{l: l \in r_i} p_l(c_l, \lambda^l) = (\lambda_i - \mu_i)/\lambda_i$, where $\lambda^l \doteq \sum_{i: l \in r_i} \lambda_i$ is the aggregate input rate at link l. We have

Lemma 1: $U(\overrightarrow{\lambda})$ is strictly concave, and has a unique interior maximum denoted as U_{max}.

Proof. Logarithmic function is strictly concave. Every other component in $U(\overrightarrow{\lambda})$ is also concave. The definition domain $\overrightarrow{\lambda} > 0$ is a convex set. Plus $\forall i, \lim_{\lambda_i \to 0} \partial U(\overrightarrow{\lambda})/\partial \lambda_i > 0$ and $\lim_{\lambda_i \to \infty} \partial U(\overrightarrow{\lambda})/\partial \lambda_i < 0$ we get this result. □

Theorem 1: $V(\overrightarrow{\lambda}) \triangleq U_{max} - U(\overrightarrow{\lambda})$ is a Lyaponov function for the system (16) which is stable.

Proof. Apparently $V(\overrightarrow{\lambda}) \geq 0$. We have

$$\frac{\partial V}{\partial \lambda_i} = -\frac{\partial U}{\partial \lambda_i} = -\frac{\dot{\lambda}_i}{m_i \lambda_i}$$

and then

$$\frac{dV}{dt} = \sum_{i \in I} \frac{\partial V}{\partial \lambda_i} \cdot \frac{d\lambda_i}{dt} = -\sum_{i \in I} \frac{\dot{\lambda}_i^2}{m_i \lambda_i} \leq 0. \quad \square$$

Corollary 1: $U(\overrightarrow{\lambda})$ is maximized at the stable point.

9.1.2 Fairness

Let's define flow i's net benefit

$$B_i(\lambda_i) \triangleq w_i \log \lambda_i - s_i \int_0^{\lambda_i} \frac{x - \beta_i \mu_i}{x}\, dx \tag{19}$$

which includes a utility and a penalty function. We get

Theorem 2: The maximization of $U(\overrightarrow{\lambda})$ is the same as a set of games which maximizing $B_i(\lambda_i)$, namely

$$\max_{\overrightarrow{\lambda} > 0} U(\overrightarrow{\lambda}) \;\Leftrightarrow\; \max_{\lambda_i > 0} B_i(\lambda_i), \forall i \in I.$$

Proof. Note $\forall i$, we have $\partial U(\overrightarrow{\lambda})/\partial \lambda_i = \partial B_i(\lambda_i)/\partial \lambda_i$, plus Lemma 1 and $\partial^2 B_i(\lambda_i)/\partial \lambda_i^2 \leq 0$ we obtain this. □

This result shows all flows' independently maximizing their own net benefits effectively maximizes a global interest. Now let's see what kind of fairness it leads to. Under ideal condition, β_i could be set to almost 1. If all flows compete network resources so aggressively that congestion probabilities are very large, we have

Lemma 2: If $\beta_i = 1$, $p_i(k) \to 1$, and flow i's utility function

$$U_i(\lambda_i) \triangleq w_i \log \lambda_i, \tag{20}$$

then a constrained convex optimization is lead:

$$\max_{\overrightarrow{\lambda} > 0} U(\overrightarrow{\lambda}) \;\Rightarrow\; \max_{\lambda_i > 0} \sum_{i \in I} U_i(\lambda_i) \;\; st. \sum_{i: l \in r_i} \lambda_i \leq c_l, \;\forall l \in L.$$

Proof. We get $s_i \to \infty$ thus $p_l(c_l, \lambda^l)$ has to be 0. According to (18) we have $\lambda^l \leq c_l, \;\forall l$. □

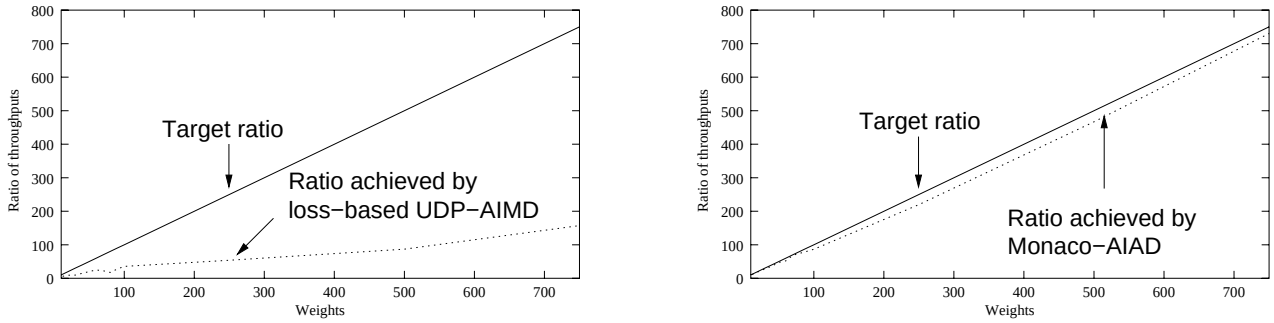


Figure 9: Service Differentiation Capability

Theorem 3: Rate allocation at the stable point of the system (16) is weighted proportionally fair [17].

Proof. Due to Corollary 1 and Lemma 2, since flow i's utility function $U_i(\lambda_i)$ is strictly concave, according to nonlinear programming optimality condition [4], if $\hat{\lambda}_i$ is the allocation to flow i associated with the system stable point, then for any other allocation λ_i, we have

$$\sum_{i \in I} U_i'(\hat{\lambda}_i) \cdot (\lambda_i - \hat{\lambda}_i) = \sum_{i \in I} w_i \cdot \frac{\lambda_i - \hat{\lambda}_i}{\hat{\lambda}_i} < 0.$$

It is the definition of weighted proportional fairness. □

9.1.3 Queue Bound

Theorem 4: Any core router inside the network controlled by (16) is with bounded queue.

Proof. Every core router is shared by a finite number of flows. Consider a core router which is the bottleneck of an arbitrary flow i, we just need to prove i's contribution to the router's queue is bounded. Assume network status changes slowly such that during several control cycles the available bandwidth μ_i doesn't change. According to algorithm (8), flow i increases its ingress rate from $\lambda_{i,0}, \cdots, \lambda_{i,k-1}$ to $\lambda_{i,k}$, each step by α_i, and then decrease the rate from $\lambda_{i,k}$ after detecting congestion. The correspondent accumulations are $a_{i,0}, \cdots, a_{i,k-1}, a_{i,k}$. Note the effect of an ingress rate is measured d_i^f length of time later at the egress router and, again, it takes d_i^b length of time to inform the ingress router. We have

$$0 \le a_{i,0}, \cdots, a_{i,k-2}, a_{i,k-1} < \epsilon_i \le a_{i,k}$$

and

$$a_{i,k-1} - a_{i,k-2} = (\lambda_{i,k-2} - \mu_i) \cdot d_i^r + \alpha_i \cdot d_i^f,$$

thus

$$\lambda_{i,k-2} - \mu_i < \frac{\epsilon_i - \alpha_i d_i^f}{d_i^r}.$$

The maximal queue for flow i is achieved d_i^b length of time after $\lambda_{i,k}$ takes effect, i.e.,

$$\begin{aligned} max(a_i) &= a_{i,k-1} + (\lambda_{i,k-1} - \mu_i) \cdot d_i^b + (\lambda_{i,k} - \mu_i) \cdot d_i^r \\ &= a_{i,k-1} + (\lambda_{i,k-1} - \mu_i) \cdot (d_i^b + d_i^r) + \alpha_i d_i^r \\ &< \epsilon_i + (\frac{\epsilon_i - \alpha_i d_i^f}{d_i^r} + \alpha_i) \cdot (d_i^b + d_i^r) + \alpha_i d_i^r \\ &< \epsilon_i + (\frac{\epsilon_i}{d_i^r} + \alpha_i) \cdot 2\, d_i^r + \alpha_i d_i^r \\ &= 3\,(\epsilon_i + \alpha_i d_i^r). \end{aligned}$$

So the queue bound for the output link ℓ of the router is

$$Q_\ell \le \sum_{i:\ell \in r_i} max(a_i) < \sum_{i:\ell \in r_i} 3\,(\epsilon_i + \alpha_i d_i^r). \quad \square$$

9.2 Service Example

This section presents a simple weighted service differentiation example and demonstrates that the EC-framework based upon Monaco allows the target differentiation (ratio of throughputs of two flows) to be achieved for a large range 800:1 (see Figure 9), well beyond what has been achieved with existing loss-based differentiation (approximately 100:1 in [25] and 10:1 in [10]). We achieve this ratio by setting each flow's target accumulation proportional to its weight, with the fundamental consequence that large weight results in proportionally large queue. The loss-based simulation was done using AIMD algorithm with weighted increase parameter differentiation.

Though we use only a simple loss-based model, other authors [10, 25] have studied this model with carefully designed schemes, and still they see the limitations of the loss-based model. For more general service differentiation (e.g., bandwidth guarantees etc.), the loss-based model is less suitable because loss interacts with end-to-end transport mechanisms and is very hard to manage effectively without stateful AQM schemes available at bottlenecks.