An Accumulation-based, Closed-loop Scheme for Expected Minimum Rate and Weighted Rate Services

David Harrison†, Yong Xia, Shivkumar Kalyanaraman, Arvind Venkatesan‡
†EECS Department, University of California, Berkeley, CA 94720, USA
ECSE and ‡CS Departments, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
Contact E-mail: shivkuma@ecse.rpi.edu

***Abstract*— Traditionally QoS capabilities have been constructed out of open-loop building blocks such as packet schedulers and traffic conditioners. In this paper[1], we consider *closed-loop* techniques to achieve a range of service differentiation capabilities. We use an accumulation-based congestion control scheme [31] as a data-plane building block to provide an *expected minimum rate* service which is similar to Frame Relay CIR/PIR and DiffServ assured service and a *weighted rate* service which achieves a significantly larger range than the loss-based approaches using TCP [9] [24]. The central theme is to allocate router buffer space among competing flows in a *distributed* manner to meet the above rate differentiation objectives. Because both services are provided using congestion control mechanisms, they are meaningful in steady state and can be modelled as moving the equilibrium in Kelly's nonlinear optimization framework [16]. This scheme needs no admission control; Instead, during oversubscription it degrades to a well-defined, policy-controlled bandwidth allocation. It does not require Active Queue Management (AQM) at bottlenecks. However, with AQM, we achieve near zero queue with high utilization. We use ns-2 simulations and Linux kernel implementation experiments to demonstrate the performance of the services.**

I. Introduction

As the Internet evolves as a telecommunication infrastructure, its best-effort service model must be augmented to provide quality of service (QoS) to support more application requirements. For example, researchers have developed the Integrated Service (IntServ) architecture to provide bandwidth and delay guaranteed services and the Differentiated Service (DiffServ) architecture for service differentiation. Both architectures place their service building blocks in network routers and thus hurdled by the incremental update of the Internet. Trying to move functionalities out of network and place them into end hosts, this paper proposes the use of accumulation-based, closed-loop congestion control mechanisms as a data-plane building block to provide expected minimum rate and weighted rate services.

The Expected Minimum Rate (EMR) refers to a service that offers a minimum contracted rate assurance plus a proportionally fair share of the remaining available capacity. It is conceptually similar to the Frame Relay CIR/PIR service that guarantees (through admission control) a Committed Information Rate (CIR) and can burst up to a Peak Information Rate (PIR) [12]. EMR service is also similar to the DiffServ assured service [3], though the latter has multiple classes and drop precedences within each class. Both DiffServ assured service and Frame Relay CIR/PIR service are realized using *open-loop* building blocks, such as packet scheduling and traffic conditioning [10] [26] [13] [30].

The weighted rate service (WRS) building block provides a weighted bandwidth differentiation among flows that share the same path. We demonstrate the significantly wider achievable range of weights comparing to the loss-based closed-loop weighted differentiation approaches using TCP [9] [24].

We develop concrete service building blocks based upon our prior work on a family of Accumulation-based Congestion Control (ACC) schemes that use accumulation, the number of buffered packets of a flow inside network, as a congestion measure [31]. An ACC control loop measures its accumulation and adjusts its sending window to keep the accumulation around a target value. In [31] we proved that ACC is globally stable, achieves weighted proportional fairness, and has an equilibrium decided by the target accumulations. Our service building blocks map a target rate allocation onto target accumulations. One key concern is that ACC over a network of FIFO queues inherently implies that the

[1]This work was supported in part by NSF under contracts ANI-9806660 and ANI-9819112, by DARPA under contract F30602-00-2-0537, and a grant from Intel Corp. David Harrison was with Rensselaer Polytechnic Institute when he did this work.

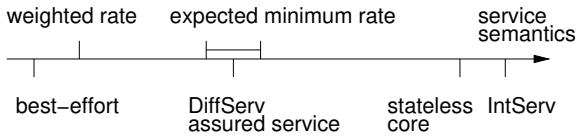


Fig. 1. Network QoS spectrum illustrates a range of network QoS from best-effort service, differentiated service to rate and delay guaranteed service. Weighted rate service and expected minimum rate service locate between the best-effort and differentiated services.

network builds physical queue, and thus the range of services is limited by buffer sizes within the network. We overcome this constraint by leveraging an Active Queue Management (AQM) mechanism that communicates the virtual queueing delay [7] [31] incurred on a virtual queue [19]. With such an AQM mechanism, we achieve near-zero queue and high utilization at routers.

Figure 1 illustrates a *qualitative* spectrum of packet-switching network QoS capabilities ranging from best-effort service controlled by TCP [15], DiffServ [3], to rate/delay guaranteed service offered by IntServ [4], including frameworks such as Stateless Core (SCORE) [28]. Services on the right-hand side of the diagram are more complex and offered at finer granularity. The left side of the spectrum implies less quantitative services, the use of FIFO queuing, closed-loop congestion control and potentially AQM mechanisms inside the network. Our framework resides in the middle of the spectrum, to the left of DiffServ assured service in the sense that we do not impose the complexity required to implement drop precedence inside the network, but to the right of DiffServ in the sense of the wide service range achieved. Unlike IntServ and SCORE that offer per-packet assurances, our service semantics are meaningful only in *steady state* due to the use of closed-loop control. In other words, our model offers the lower end of the service spectrum at very low complexity.

We evaluate the EMR and WRS building blocks using ns-2 [25] simulations and Linux kernel v2.2.18 implementation experiments. Several fundamental issues such as graceful service degradation (when there is oversubscription, limits on accumulation, or finite buffer size), and trade-offs (like non-AQM droptail vs. optional AQM support at bottlenecks) are explored. It is important to note that this paper only develops the abstract service model, albeit with ns-2 simulation and Linux implementation validation, and does not explore architectural issues such as multi-ISP service coordination, etc. However, we remark that the abstract model developed in this paper offers an attractive potential of *overlaying* QoS capabilities on top of an existing FIFO network from network edges.

This paper is organized as follows. In Section II we review related work. In Section III we describe ACC fluid model and one of the ACC schemes, Monaco, that serve as the basis on which we provide the EMR service building block in Section IV and the WRS service building block in Section V. A set of ns-2 simulation results are given in Sections IV-C and V-B. In Section VI we multiplex all the services to show that they can co-exist in a complex network. Section VII concludes this paper.

II. Related Research

IntServ architecture [4] was proposed to provide end-to-end bandwidth and delay guarantees. But there is no real deployment seen due to its algorithmic complexity and that it has to maintain per-flow state inside routers. By carrying the per-flow state in packet header and thus eliminating them from inside routers, SCORE [28] is a significant step toward achieving guaranteed service while keeping the core Internet stateless.

DiffServ [3] was developed to implement scalable service differentiation in the Internet. It defines an assured forwarding service which provides delivery of IP packets in four independently forwarded classes with three different levels of drop precedence. Wang suggested a user-share differentiation mechanism [30].

Clark and Fang proposed that each flow uses a "profile" specifying how much capacity should be allocated to TCP connections during congestion [8]. A component is placed at the edge of the network that marks a bit in passing packet headers to denote whether packets are in- or out-of-profile. The bottleneck then uses two RED algorithms [11] in parallel to drop packets randomly during congestion with preference toward keeping packets that are marked in-profile.

All the proposals discussed above are basically open-loop approaches. Researchers have also designed closed-loop mechanisms to provide service differentiation. For instance, Crowcroft et al. implemented MulTCP which makes a TCP connection behave as w TCP connections by increasing the congestion window of the single TCP connection by w packets in each RTT and, when a loss occurs, decreasing it by only $\frac{w-0.5}{w}$ [9]. Nandagopal et al. provided a systematic analysis on how to adjust TCP Reno congestion control increase/decrease parameters to achieve the weighted bandwidth differentiation [24]. In both work the service is achieved by (adaptively) changing congestion control increase/decrease parameters. We take a different approach to achieve service differentiation by manipulating accumulation allocation among competing flows.

A different line of research started from Kelly's optimization framework [16], followed by Low et al. [21] and Kunniyur et al. [18], where they model network congestion control as a nonlinear optimization problem under which all users try to maximize their own interest, subject to a set of capacity constraints. Our scheme fits into this optimization framework. Kunniyur and Srikant developed an Adaptive Virtual Queue (AVQ) algorithm [19] which we leverage to keep a small physical queue in congestion routers (see Section IV-A). Low et al. proposed an optimization model [22] for TCP Vegas [5] and suggested to improve Vegas performance by using a buffer management mechanism called Randomly Exponential Marking (REM) [1]. All these work were developed for the purpose of congestion control. In this paper, we extend ACC, Monaco+AVQ in particular, as a data-plane building block to provide service differentiation.

III. Background

In this section we briefly describe our prior work on the ACC fluid model and its packet-switching network implementation, the Monaco scheme. Monaco solves the technical problems of Vegas and thus serves as a basis for the work in this paper. For the complete description of ACC, Vegas and Monaco, the reader is referred to [31].

A. ACC Fluid Model

We define the accumulation concept using a bit-by-bit fluid model. Then we develop a general congestion control algorithm based on accumulation and show its steady state and dynamic properties.

1) Accumulation: Consider an ordered sequence of FIFO nodes $\{R_1, \ldots, R_J\}$ along the path of a unidirectional flow i. The flow comes into the ingress node R_1 and, after passing some intermediate nodes $R_2, \ldots, R_{J-1}$, goes out from the egress node R_J[2]. The propagation delay from node R_j to node R_{j+1} is a constant d_j.

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

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

where $q_{ij}(t - \sum_{k=j}^{J-1} d_k)$ is flow i's bits queued in node R_j at time $t - \sum_{k=j}^{J-1} d_k$. Note it includes only those bits backlogged inside node buffers, not those stored on transmission links. This definition serves as a reference to implement an unbiased accumulation estimator in Section III-B. We aim to control flow rate by controlling its accumulation.

2) Control Algorithm: We apply a window-based control to use accumulation as a measure to detect network congestion. If accumulation is low, we increase congestion window; otherwise, we decrease it to drain accumulation. Specifically, we try to maintain a constant target accumulation a_i for each flow i using a general ACC control algorithm:

$$\dot{w}_i(t) = -\kappa \cdot f(a_i(t) - a_i) \tag{2}$$

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

Obviously the above equation includes a *set* of algorithms. The reason we present a general algorithm is that *all instance algorithms which fit into Equation (2) share a common steady state property of proportional fairness*, as shown in the next subsection. By choosing different f functions, we can instantiate the above general algorithm into a set of control policies including the additive-increase-additive-decrease (AIAD) policy [6] and a proportional control policy used by Monaco. Thus to achieve a particular rate allocation, we have the freedom to choose from a set of control policies. In this sense, the ACC congestion control is decoupled into two different issues: setting a target steady state allocation (fairness) and then designing a control policy (stability and dynamics) to achieve that allocation.

3) Properties: Congestion control can be formalized as a resource allocation problem [16] [21] [18] [23]. Using steady state queuing analysis and nonlinear optimization we proved in [31] that the above ACC control algorithm is globally stable and drives the network to an equilibrium of weighted proportional fairness, with target accumulation a_i as flow i's weight. *It is this weight a_i that enables the capability of service differentiation*. In principle, since the equilibrium is decided by all a_i's, we can accordingly adjust a_i to steer the equilibrium in order to achieve a specific service, e.g., EMR or WRS rate allocations discussed in Sections IV and V. This decoupling allows us to avoid the major pitfalls in the loss-based service differentiation approaches, as discussed in Section V-C.

[2]In this paper we map R_1/R_J nodes as source/destination end hosts to form an end-to-end control loop. R_1/R_J can also be mapped as ngress/egress edge routers. However, we focus on data-plane functions in this paper and neglect architecture issues.

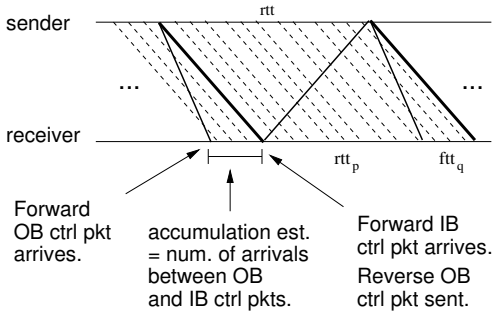


Fig. 2. Monaco accumulation estimator generates a pair of back-to-back OB and IB control packets once per RTT at the sender. The number of data packets at the receiver between the OB and IB control packets is measured as accumulation, which is sent back to the sender using an OB control packet.

B. Monaco Scheme

Based on the measurement and control of accumulation, we have shown that a family of congestion control schemes can be derived [31]. One example is TCP Vegas which estimates accumulation as $sending_rate \times (rtt - rtt_p)$ at sender side, where rtt is round trip time (RTT) and rtt_p refers to round trip propagation delay. The Vegas accumulation estimator is sensitive to both the measurement error of the propagation delay and reverse path congestion. Therefore we designed a receiver-based, out-of-band Monaco scheme that ensures robust accumulation estimation [31].

1) Accumulation Estimation: Monaco estimates accumulation at the *receiver* side. It generates a pair of back-to-back control packets once per RTT at the sender 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 propagation delay in the forward path. The IB control packet goes along with regular data packets and reaches the receiver after experiencing the current queueing delay in the network. The time interval between the OB and IB control packets measured at the receiver samples the sum of the queueing delays in the forward path. Considering a network in steady-state with enough buffers where there is no packet loss, then by Little's law, the average number $\hat{a}_M$ of data packet arrivals at the receiver after the OB control packet, but before the IB control packet equals the average accumulation.

Obviously this comes with an extra requirement of two separate FIFO queues inside network routers, with a high priority queue for OB control packets and low priority queue for IB control and data packets.

2) Control Policy: Monaco employs a proportional control policy. It tries to maintain a constant target accumulation a_i for each flow i by adjusting its congestion window according to:

$$cwnd_i(n+1) = cwnd_i(n) - \kappa \cdot (\hat{a}_{iM} - a_i) \quad (3)$$

where $\hat{a}_{iM}$ is the accumulation estimate, κ is set to 0.5, and $cwnd_i(n)$ is the congestion window value at a control period n. In addition we conservatively bound the increase to within one MTU per RTT.

Monaco sender implements rate-based pacing to smooth incoming traffic into the network using a token bucket shaper with a rate value of $cwnd\,/\,rtt$ to alleviate traffic burstiness. It also includes reliability enhancements for control and data packets lost.

IV. Expected Minimum Rate Service

In this section we demonstrate the EMR service building block. We aim to provide any flow *a contracted (or expected minimum) bandwidth plus a proportional share of remaining capacity*. It is achieved by keeping an appropriate amount of accumulation inside network for that flow. We mean "expected" in the sense that the minimum can be satisfied if there is no oversubscription. If there is, obviously we can not fulfill all the demands. Therefore a graceful service degradation is defined. We show theoretic result for a general network topology and evaluate it using a set of ns-2 simulations based on Monaco.

A. Theory

Let's consider a network of a set $L = \{1, \ldots, L\}$ of links, shared by a set $I = \{1, \ldots, I\}$ of flows. Each link $l \in L$ has capacity c_l. Flow $i \in I$ passes a route L_i consisting of a subset of links, i.e., $L_i = \{l \in L \mid i \text{ traverses } l\}$. A link l is shared by a subset I_l of flows where $I_l = \{i \in I \mid i \text{ traverses } l\}$.

As we discuss in Section III-A.3, a specific rate allocation corresponds to a particular equilibrium of the control algorithm (2); And the equilibrium is decided by the target accumulation a_i. So service provisioning is mapped to the management of the target accumulations. We use the following steady state analysis to introduce our algorithm.

Suppose we provide flow i a bandwidth x_i^* which is the sum of an EMR x_{ie} and a proportional share x_{ip}^* of the remaining capacity[3]:

$$x_i^* = x_{ie} + x_{ip}^*. \quad (4)$$

[3]It will be clear very soon why we use symbols x_i^* and x_{ip}^* instead of x_i and x_{ip} here and why we call x_{ip}^* a proportional share.

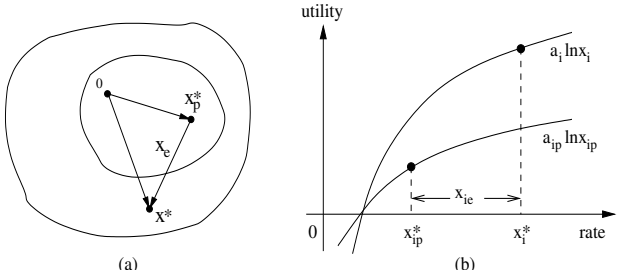


Fig. 3. EMR could be modelled as two simultaneous nonlinear optimization problems: (a) Convex constraint sets showing $\vec{x^*} = \vec{x_e} + \vec{x_p^*}$: inner one for x_{ip} while outer one for x_i; (b) Logarithmic utility functions of x_i and x_{ip}.

We achieve this allocation by keeping for flow i a total accumulation a_i which also includes correspondingly two parts: a_{ie} for x_{ie} and a_{ip} for x^*_{ip}, i.e.,

$$a_i = a_{ie} + a_{ip}. \tag{5}$$

According to Little's law, we have

$$a_i = x^*_i \cdot t_{iq}, \tag{6}$$
$$a_{ie} = x_{ie} \cdot t_{iq}, \tag{7}$$
$$a_{ip} = x^*_{ip} \cdot t_{iq} \tag{8}$$

where t_{iq} is the steady state queueing delay experienced by flow i in its forward path. Equations (5)-(8) lead to

$$a_{ip} = (1 - \frac{x_{ie}}{x^*_i}) \cdot a_i. \tag{9}$$

where x_{ie} is the expected rate known *a priori*, x^*_i can be measured at receiver side, and a_i is provided by the accumulation estimator.

Obviously, if we keep constant accumulation for a flow, then its (proportionally fairly) allocated rate is decided by network routing and other competing flows. On the contrary, if we want to keep a constant rate for a flow, independent of the changing environment, then the corresponding accumulation has to be adaptively set. This is reflected by t_{iq} in Equation (7). To avoid setting a varying parameter a_{ie}, or equivalently a_i in Equation (6), we use Equation (9) to calculate a_{ip} which automatically accounts for a_{ie}. Then we just need to keep a_{ip} constant

Firstly let's assume enough buffers are provided in all congested routers and there is no oversubscription. We consider the total rate x^*_i achieved with corresponding accumulation a_i. As proved in [31][4], the flow rate x^*_i is the unique maximum of the following nonlinear optimization problem related to a_i:

$$maximize \quad \sum_{i \in I} a_i \, lnx_i \tag{10}$$
$$subject\ to \quad \sum_{i \in I_l} x_i \le c_l, \ \forall l \in L \tag{11}$$
$$x_i > 0, \ \forall i \in I$$

meaning that x^*_i achieves weighted proportional fairness:

$$\sum_{i \in I} a_i \cdot \frac{x_i - x^*_i}{x^*_i} \le 0 \tag{12}$$

where x_i is any feasible rate satisfying constraint (11).

Beyond the expected rate x_{ie}, we then consider the rate x^*_{ip} achieved with corresponding accumulation a_{ip}. Similarly we can prove that the flow rate x^*_{ip} is the unique maximum of the sub-problem related to a_{ip}:

$$maximize \quad \sum_{i \in I} a_{ip} \, lnx_{ip} \tag{13}$$
$$subject\ to \quad \sum_{i \in I_l} x_{ip} \le c_l - \sum_{i \in I_l} x_{ie}, \ \forall l \in L \tag{14}$$
$$x_{ip} > 0, \ \forall i \in I$$

meaning x^*_{ip} achieves similar proportional fairness:

$$\sum_{i \in I} a_{ip} \cdot \frac{x_{ip} - x^*_{ip}}{x^*_{ip}} \le 0 \tag{15}$$

where x_{ip} is any feasible rate satisfying constraint (14).

So a system providing an EMR service actually does two nonlinear optimization problems; but these two problems are not independent – they are *simultaneous* in that as long as one problem achieves its optimality, the other also achieves its optimality at the same time. We illustrate this result in Figure 3. The reader is referred to Section 5.3 of [14] for an analysis regarding general utility functions.

If the assumptions do not hold, i.e., there is either oversubscription or underprovisioned buffers, then the above two optimalities might not actually be realized in a practical system. There are three boundary conditions affecting the realizability of EMR:

- Oversubscription: Without admission control, the constraint (14) might be always invalid. Or it is possible that

$$\sum_{i \in I_l} x_{ie} \ge c_l, \ \exists \, l \in L; \tag{16}$$

- Accumulation limit: As we keep for each flow a steady state accumulation, we want to put an upper-bound A_i which limits the queuing delay introduced, namely,

$$a_i = \sum_{L \in L_i} q_{il} \le A_i, \ \forall i \in I; \tag{17}$$

[4]For the reviewers' convenience we provide the brief proof in Appendix. See http://www.rpi.edu/~xiay/pub/acc.ps.gz for the complete version.

- Buffer overflow: Even with the above accumulation limit, we can not yet guarantee that each router buffer Q_l is sufficiently provisioned, i.e.,

$$q_l = \sum_{i \in I_l} q_{il} \leq Q_l, \ \forall l \in L. \tag{18}$$

When any of the boundary conditions is effective, EMR can not be fulfilled. Therefore we need to define a degraded service for this case.

Firstly, we argue that the constraints (16) and (18) can be translated equivalently into the form of (17). For (18), it is obvious because both (17) and (18) are constraints on q_{il}. For (16), the intuition is that when there is an oversubscribing flow, then if its demand were satisfied, it would incur *infinite* accumulation, which means before this happens, its accumulation constraint (17) will be effective. So these three constraints can be summarized into one form of (17) with an equivalent accumulation limit A_i^{eq}. To take into account all the above boundary conditions (16)$\sim$(18), we define a new optimization problem similar to (10) but with changed coefficients a_i' and thus different, even still logarithmal, utility functions:

$$\begin{aligned} maximize \quad & \sum_{i \in I} a_i' \, ln x_i' \\ subject \ to \quad & \sum_{i \in I_l} x_i' \leq c_l, \ \forall l \in L \\ & x_i' > 0, \ \forall i \in I \end{aligned} \tag{19}$$

where

$$a_i' = min(a_i, A_i^{eq}), \ \forall i \in I. \tag{20}$$

Again, its optimal allocation $x_i^{'*}$ achieves weighted proportional fairness:

$$\sum_{i \in I} a_i' \cdot \frac{x_i' - x_i^{'*}}{x_i^{'*}} \leq 0 \tag{21}$$

where x_i' is any feasible rate.

Consequently, when there is either oversubscription, accumulation limit, or underprovisioned buffers, these conditions affect the realization of (10) and (13), i.e., EMR can not be provisioned. A new optimization problem (19) is automatically defined by adding these constraints (16)$\sim$(18) into (10), achieving a weighted proportional rate allocation which can be computed from the boundary conditions. The weight is changed from a_i to a_i' defined by Equation (20). As a result, EMR gracefully degrades into a weighted proportionally fair rate, with a weight a_i'. Further, since we have the freedom to set the accumulation limit A_i for each flow i in (17), this parameter provides a policy-based control on bandwidth allocation. We use simulations to illustrate this in the next subsection.

Now let's look at a trade-off. Even if we set an accumulation limit for each flow, the steady state queuing delay or physical queue length might be large as the number of flows increases. Is it possible to provide the requested services based on managing accumulation and, at the same time, keep a small steady state queue length? We have adopted the AVQ algorithm [19] which emulates an adaptively changing link capacity such that the steady state queue length is sufficiently small. We compute a virtual queueing delay [7] defined as the ratio of virtual queue length divided by virtual capacity and add it into the forward control packet provided by the out-of-band accumulation estimator[5]. We call this mechanism Adaptive Virtual Delay (AVD) algorithm. A nice property of AVD is that it is incrementally deployable since a mixed set of non-AQM droptail and AVD routers can work together (see Section VI and [31]). In such an environment the accumulation estimate will be $\hat{a} = \hat{a}_{FIFO} + x \cdot \hat{t}_{VD}$ where $\hat{a}_{FIFO}$ is accumulation in those FIFO routers, x is the received rate and $\hat{t}_{VD}$ is the sum of all virtual delays at those AVD routers.

To sum up, we use the algorithm below to provide the EMR service. It includes seven steps.

Algorithm 1 Expected Service Pseudo-code at Sender

cwnd = the congestion window in bytes
pwnd = the congestion window in the previous RTT
ssthresh = the slow start threshold
srtt = the smoothed RTT estimation
A = the total accumulation limit
ε = the target accumulation beyond the EMR
a = accumulation estimate

(1) $x = pwnd * 8.0/srtt$;
(2) $a_p = max(a * \ (1 - x_e/x), \ 0.0)$;
(3) $pwnd = min(pwnd + mtu, \ cwnd)$;
(4) $cwnd = pwnd - k * max(a_p - \varepsilon, \ a - A)$;
(5) if $(a > A \ || \ a_p > \varepsilon)$ { $ssthresh = cwnd$; }
(6) else {
  (6.1) if $(pwnd + mtu >= ssthresh)$
    $ssthresh = cwnd$;
  (6.2) $cwnd = min(pwnd * 2.0, \ ssthresh)$; }
(7) rate_limit = $cwnd * 8.0/srtt$;

In Step 1, we compute the departure rate x as the

[5]This accumulation estimator (see Section III-A.1) sends a control packet once per RTT out-of-band to avoid standing queues that affect the Vegas accumulation estimator. If no control packet is available from the accumulation estimator then the virtual delay may be communicated using bit marking as in REM [1].

bits transmitted in the last RTT divided by the smoothed RTT, or $srtt$.

In Step 2, we compute the accumulation a_p incurred beyond the EMR according to Equation (9). When a control loop is ramping up or during oversubscription, the departure rate x may be less than the EMR x_e causing a_p to be negative. So we max with 0.0 to force nonnegativity.

In Step 3, we force a $pwnd$ that is within 1 MTU of $cwnd$ to be equal to $cwnd$. This is necessary because our algorithm stops sending when the packet at the head of the queue would cause $pwnd$ to exceed $cwnd$.

In Step 4 we set the congestion window according to the Monaco control policy defined in Equation (3) and the accumulation limit A.

In Steps 5–6, we determine whether step 4 was a decrease or an increase step. In Step 5, we stop a slow start by reducing $ssthresh$ to the current congestion window size. In other words, each control loop slow starts until congestion is first detected. Step 6.1 ensures that when a $cwnd$ decreases due to something other than congestion (e.g., decreasing user demand), $ssthresh$ does not decrease, i.e., $ssthresh$ tracks the congestion level in the network rather than the user demand. It bounds the increase step to one MTU per RTT. Thus Monaco increases no more aggressively than TCP's additive-increase-multiplicative-decrease (AIMD) and less aggressively than TCP in the neighborhood of the equilibrium. Though not shown here, Monaco backs off by one half in response to loss and thus Monaco degrades to TCP in the presence of significant loss. Step 6.2 bounds slow start by $ssthresh$.

Step 7 sets the rate on the token bucket shaper used in rate-based pacing.

B. How to Set Accumulation Limit

One question remaining is how to set the accumulation limit A_i, which serves as a policy-controlled parameter for an EMR flow i.

By the definition in Equation (1), the accumulation of a flow is distributed in the form of buffered packets among the congested routers on its path. According to Equation (5), to achieve EMR, accumulation a_i includes two parts: a_{ip} should equal to three packets, as we set for a general non-EMR flow; a_{ie} should adapt according to the changing environment. Since A_i is an upper bound of a_i, it should be set in proportion to the number of congested routers and how congested they are. On the other hand, keeping a larger A_i means to give flow i more chance to get its EMR under oversubscription; A smaller A_i means its EMR will probably not be obtained in the same situation. We use concrete simulation examples in the next subsection to further demonstrate this.

C. Simulations

We evaluate the performance of the EMR service building block using ns-2 simulations on a single bottleneck. Each flow has a different propagation delay. In Section VI we provide ns-2 simulation and Linux implementation experiment results for a more complex network of multiple bottlenecks. In all experiments we set data packet size to 1KB.

We simulate a single 100Mbps bottleneck with 4ms propagation delay shared by ten flows using Monaco. We use simulation because this allows us to temporarily set aside the practical limitation of finite buffer sizes, so that we can study the range of services in the absence of packet loss. Flow 0 has an EMR x_{1e} while other nine flows request a proportional rate service (i.e., with an EMR of 0). Each source i ($0 \leq i \leq 9$) is connected to the bottleneck via an 1Gbps link with one-way propagation delay $11i+1$ms. We performed two kinds of simulations, one using non-AQM droptail bottleneck, the other using AVD bottleneck.

In the first set of simulations we evaluate the range of satisfiable EMRs without AQM and we demonstrate the effects of setting the accumulation limit. For each accumulation limit of 30KB and 3000KB, we run simulations each with a different expected rate x_{1e} from 0 to 110 Mbps. All other flows have a target accumulation of 3KB. The result is shown in Figure 4(a) and zoomed in Figure 4(b). With $A = 3000$KB, we are able to allocate up to 99.1Mbps (i.e., $\frac{3000\times100}{3000+9\times3}$) of the 100M bottleneck to a single flow before the accumulation limit is reached. When $A = 30$KB, the maximum satisfiable EMR is $\frac{30\times100}{30+9\times3} = 52.6$Mbps, demonstrating that the unsatisfied EMR due to the constraint (17) degrades to a weighted proportionally fair rate.

The upper part of Figure 4(d) shows that the queue length grows dramatically (note the logarithmic scale of the vertical axis) as the EMR x_{1e} increases. This is most apparently shown by the case when $A = 3000$KB. However in all cases, the queue growth flattens when the accumulation limit is reached. The rapid queue growth as the EMR approaches available capacity demonstrates the bound on achievable services as a function of the buffer size in the network.

In the second set of simulations, we repeated the above simulations with AVD bottleneck. The range of achieved EMRs is similar to the case without AQM, as shown in Figure 4(c) for the EMR of 90-100Mbps. However, AVD keeps the queue length at near zero as

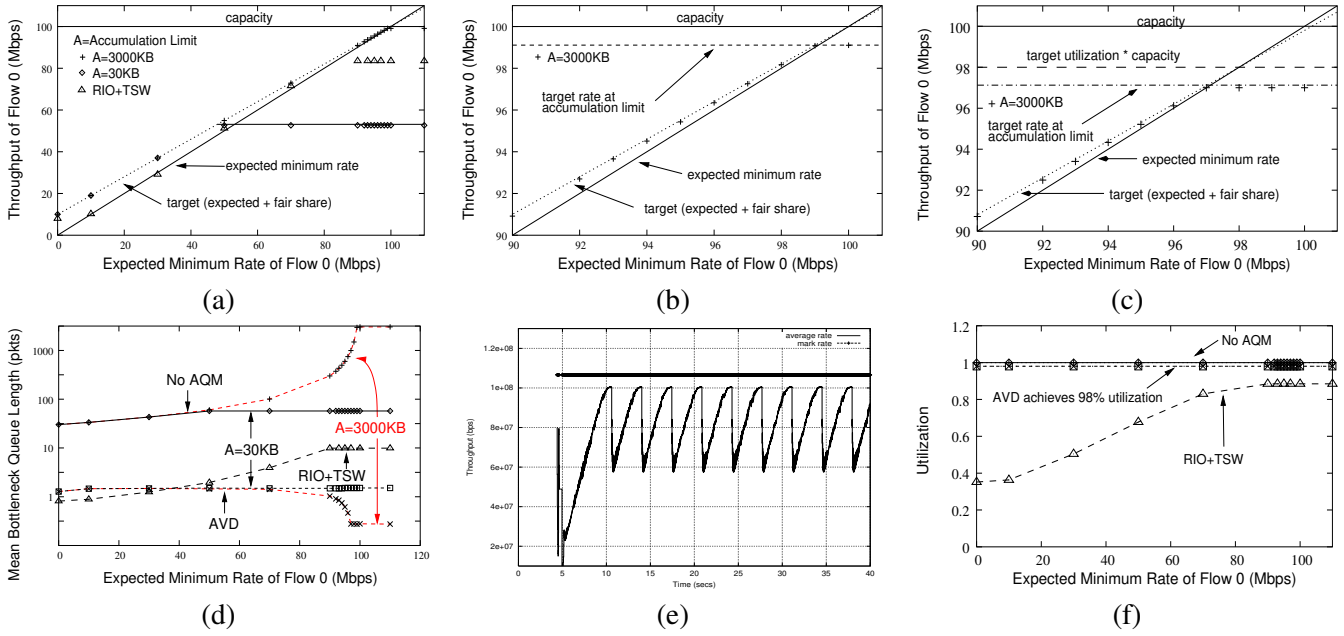


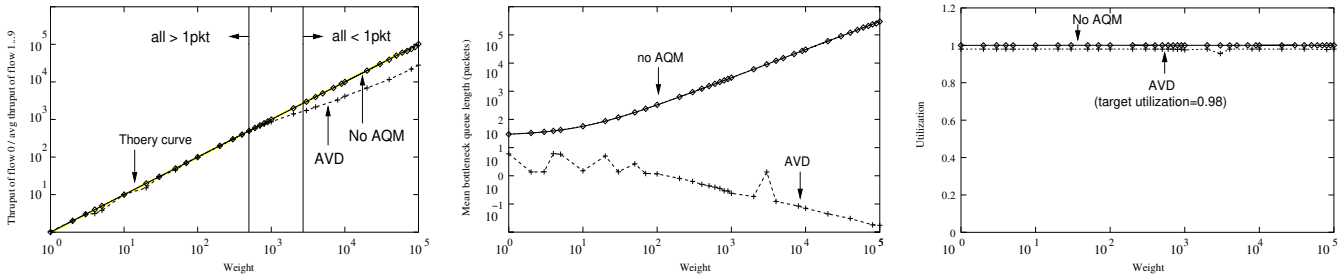
Fig. 4. EMR in a single bottleneck: (a) Monaco throughput x_0 vs. EMR x_{0e} with non-AQM droptail bottleneck and RIO throughput vs. DiffServ bandwidth allocation; (b) The former graph zoomed into 90-101Mbps; (c) Monaco throughput x_0 vs. EMR x_{0e} with AVD bottleneck, zoomed into 90-101Mbps; (d) Steady state queue length vs. EMR x_{0e} with droptail or AVD bottleneck (when flow 0's accumulation limit A is set to 30KB and 3000KB, respectively), and RIO+TSW average queue length vs. DiffServ bandwidth allocation; (e) RIO+TSW's measured rate for capacity allocation 80Mbps; (f) Bottleneck utilization when using Monaco and RIO+TSW. In these simulations, we set κ=0.5, target accumulation a=3KB, AVD's damping factor α=0.1 and target utilization β=0.98. For RIO+TSW simulations we use the parameters in [8].

depicted in the lower part of Figure 4(d), since instead of incurring physical accumulation, each control loop incurs virtual accumulation on a virtual queue. In fact the average queue diminishes as the ratio of the rates increases since 1) the packets in the fast flow are nearly evenly spaced due to rate-based pacing and therefore do not incur queue, and 2) the slower flows send so infrequently that they rarely perturb the queue.

As shown in Figure 4(f), the bottleneck utilization for all cases without AQM was 100% in the steady state (since the queue never drains completely), and when AVD was used, the utilization was always within half a percent of 98%, our target utilization for AVD.

Now we compare EMR and DiffServ assured service using a set of simulations. We varied the allocation for a TCP Reno connection using Clark et al.'s RIO building blocks [8]. To perform marking they proposes the Time Sliding Window (TSW) tagger which marks packets in-profile so long as the connection sends with rate less than 4/3 of its allocation specified in its profile. Because TCP backs off one half of its congestion window in response to loss, the expectation is that during congestion the TCP connection should sawtooth between 2/3 and 4/3 of its allocation, resulting in a long-term average rate near the allocation. RIO uses two RED [11] algorithms in parallel to randomly drops "in" and "out" packets in the queue. The RED algorithms are configured such that "out" packets are dropped before "in" packets. We use the same parameter settings for RIO and TSW as specified in Table 1 of [8]. Additionally our RIO algorithm has Explicit Congestion Notification (ECN) [27] turned on. Thus packets are marked for congestion (different than the in/out mark) when RIO would otherwise have dropped.

In Figure 4(e) we demonstrate one of the limitations of the TSW tagger and a more fundamental limitation of providing assured services with TCP+RIO. First a connection receives the same treatment for any allocation greater than 3/4 of the bottleneck capacity since all packets are marked in-profile. More importantly, in order to obtain an allocation that is a significant fraction of the bottleneck capacity, TCP must oscillate its window size on the same order as the bottleneck capacity. This oscillation is a consequence of TCP's AIMD algorithm and thus affects all profile marking, ECN marking, and AQM algorithms. Since the amplitude of this sawtooth grows roughly linearly with bit rate, accommodating a flow with a large bandwidth allocation implies one of the following three: large queues (see Figure 4(d)), limits on achievable allocations (as shown for bandwidth allocations above 80Mbps in Figure 4(e)), or low utilization (as shown in Figure 4(f) for bit rates below 80Mbps). This *fundamental limitation* of TCP+RIO makes it impossible to simultaneously achieve high utilization, low queue



(a) (b) (c)

Fig. 5. WRS in a single bottleneck: (a) Relative throughput ratio vs. weight; (b) Bottleneck queue vs. weight; (c) Bottleneck utilization. In these simulations, we set κ=0.5, target accumulation a_i=w_i*3KB, AVD's damping factor α=0.1 and target utilization β=0.98.

length, and a wide disparity in the allocated rates. By avoiding AIMD, Monaco with AVD simultaneously achieves all three.

V. Weighted Rate Service

In this section we propose the WRS service building block. Our goal is to provide *a weighted rate discrimination among flows along the same path, i.e., a flow with weight w should get a bandwidth w times of a flow with weight 1*. We show the steady state queuing theoretic result for a general network topology. Then we evaluate it using ns-2 simulations and show that the achieved WRS range significantly outperforms the loss-based approaches by *three orders of magnitude*.

A. Theory

Again let's consider the general network topology described in Section IV-A from a steady state queuing analysis perspective. At a steady state in any bottleneck link l where

$$\sum_{i \in I_l} x_i = c_l \tag{22}$$

we have its steady state queue length q_l including contributions from a subset of flows traversing that link as

$$q_l = \sum_{i \in I_l} q_{il} \tag{23}$$

where q_{il} is flow i's steady state queue at bottleneck l.

We use window-based congestion control, in which a window w_i bits of flow i could be stored either on transmission links (= $w_i - a_i$) or in router buffers as steady state accumulation

$$a_i = \sum_{l \in L_i} q_{il} \tag{24}$$

where L_i is the subset of bottleneck(s) which flow i traverses. If two flows i_1 and i_2 traverse the same path from a source to a destination, then they share the same set of bottleneck(s), namely,

$$L_1 = L_2. \tag{25}$$

At any shared bottleneck l employing FIFO scheduler which decides packet departure totally based on its arrival order, the flow rate allocation is proportional to the buffer allocation, i.e.,

$$\frac{x_1}{x_2} = \frac{q_{1l}}{q_{2l}}. \tag{26}$$

The above three equations lead to

$$\frac{x_1}{x_2} = \frac{\sum_{l \in L_1} q_{1l}}{\sum_{l \in L_2} q_{2l}} = \frac{a_1}{a_2} \tag{27}$$

which means that, for a general network topology, the rate allocation to flows along the same path is proportional to the accumulation allocation. Further, it is the *relative* accumulation allocation, instead of *absolute* accumulation allocation, that decides the rate differentiation.

B. Simulations

We evaluate the performance of the WRS service building block using ns-2 simulations on a single bottleneck with large enough buffer size to avoid packet loss. For a more realistic network of multiple congested links shared by flows passing through different numbers of bottlenecks, we provide ns-2 simulation and Linux implementation experiment results in Section VI.

We use the same single bottleneck network where an 100Mbps 4ms link is shared by ten flows with very heterogeneous RTTs. Flow 0 has a varying weight w while other flows have unit weight (i.e, $a_0 = wa_1 = \cdots = wa_9$). We performed two sets of simulations to evaluate the weighted service range.

In the first set of simulations we evaluate the service differentiation range without AQM. We did a set of simulation runs by changing w from 1 to 10^5 which

is at least three orders of magnitude larger than the weights (10∼100) achieved with TCP in [9] [24]. We discuss this huge difference in the next subsection. As shown by the upper curve in Figure 5(a), for the wide range of weight variation, accurate weighted sharing is achieved. This comes with a cost, though. The upper curve in Figure 5(b) shows that the steady state queue length at the bottleneck increases linearly with weight. The curvature in the mean queue length in Figure 5(b) is due to a y-offset for a weight 1 equal to the sum of the target accumulations (30KB). As with the EMR building block, the large queue incurred by service differentiation based on physical accumulation demonstrates the practical bound on service differentiation that would be achieved with finite buffer sizes.

Again, with AVD we are able to break the coupling between the notion of accumulation and real queuing. This is shown by our second set of simulations. Figure 5(b) demonstrates that AVD achieves average queue sizes less than 1 packet. This average diminishes as weight increases for the same reason that average queue diminishes as EMR increases as we already discussed in Section IV-C: control loops that send rarely also rarely perturb the queue.

As shown in Figure 5(a), for weights below 10^3, AQM achieves the desired weighted share. However, above about 10^3, the control loop with the large weight obtains less than the desired weighted share. This arises because AVD holds the loops with weight 1 at a window size of one. In this regime, the Monaco control policy is no longer defined, instead if the accumulation estimate is larger than the target and the window size is one then the control loop halves its shaper's send rate. Once the accumulation estimate reduces below the target, we reset the respective shaper to a rate of 1 packet per RTT. As the weight of source $S0$ increases to 491, the equilibrium window size of source $S1$ shrinks to 1. By the time the weight reaches 2691, the window sizes of all sources $S1{\sim}S9$ shrink to 1 packet. In order to obtain larger weighted shares, the control loops must either send slower or the queue must grow. As we know from Figure 5(b), the queue does not grow above a weight of 491 or 2691. Instead, we use rate-based pacing to halve the send rate whenever the window size reaches 1 and negative feedback arrives. When positive feedback arrives we react in the same way as before.

This handling for the regime of low rates is similar to the exponential back-off used by TCP when a timeout occurs and the immediate recovery to a rate of one packet per RTT when an acknowledgement arrives. Jumping to one packet per RTT represents an aggressive increase that biases the weighted share in favor of the control loops with smaller weights. This aberration occurring at high weights might be solved through careful redesign of the increase step used in the low rate regime.

As shown in Figure 5(c), without AQM, the utilization is 100%. With AQM the utilization settles around 98%, which is our target utilization for AVD.

C. Accumulation-based vs. Loss-based Approaches

Related research, such as [9] [24], has explored to manipulate packet loss rate of TCP Reno to provide similar rate differentiation. In this subsection we compare the mechanisms used by the loss-based approach and this paper. We argue that the accumulation-based approach is in principle superior to the loss-based one in that the former can achieve a significantly larger range of service differentiation (as shown by the simulation results) and at the same time maintain good dynamic performance.

In both work the service is achieved by (adaptively) changing congestion control increase/decrease parameters. This approach creates a dilemma: it is hard to achieve *both* service differentiation and good dynamic performance because the increase/decrease parameters of the congestion control algorithm decide its dynamic performance. For example, larger increase/decrease parameters lead to larger oscillation of congestion window. This is the reason why the short term behavior of the algorithm in [24] suffers and a long time simulation run is needed. Further, it's hard to accurately measure packet loss rate at end hosts.

We use a quite different approach based on accumulation allocation: a specific service, including the weighted one, is achieved by *moving* the equilibrium of the control algorithm in Equation (3). The control algorithm's dynamic behavior can be designed separately as long as it fits into the general ACC model (See Section III-A). This *decoupling* of the steady state equilibrium and dynamic control algorithm provides the capability to achieve the targeted service as well as good dynamic behavior.

Of course, since we need to keep a steady state physical queue (i.e., accumulation) for each flow, and q_l is limited by the physical buffer size Q_l shown in the constraint (18), bottleneck buffer size limits the range of the weighted service in practice. But this limitation is very different from that of [9] [24] where it is resulted from the coupling of steady state objective and dynamic performance.

VI. Service Multiplexing

In this section we provide ns-2 simulation and Linux kernel implementation experiment results to demonstrate that EMR and WRS can *co-exist dynamically* with bursty

TABLE I
MULTIPLE-BOTTLENECK SIMULATION PARAMETERS

flow	expected rate (x_{ije})	A limit (A_{ij})	weight (w_{ij})	start time	stop time
(0,0)	30Mbps	60KB	1	$U[0,5]$s	
(0,1)	35Mbps	75KB	1	25s	75s
(0,2)	50Mbps	60KB	1	50s	75s
(0,3)	10Mbps	15KB	1	$U[0,5]$s	
(2,0)	0Mbps		3	$U[0,5]$s	
web	0Mbps		1	100s	
other	0Mbps		1	$U[0,5]$s	

web-like traffic in a complex multiple bottleneck network including both non-AQM droptail and AVD routers.

A. Simulations

Firstly we show the simulation results for the network shown in Figure 6(a). We choose a topology with all equal capacities and put all control loops with EMRs along the multiple-bottleneck path, because these choices simplify target rate computations, which are described momentarily. There are four "long" flows passing through all bottlenecks and a set of "cross" flows each using only one bottleneck. Every bottleneck link has 100Mbps capacity and 1ms propagation delay. The source nodes $(1,0)\sim(1,3)$ are connected to R_0 with propagation delays evenly spread between 1 and 100ms, i.e., 1ms, 34ms, 67ms and 100ms, respectively. Nodes $(2,0)\sim(2,3)$ have the same delays but are connected to R_1 and likewise for nodes $(3,0)\sim(3,3)$ to R_2. As specified in Table I, each of the long flows has an EMR, and they start and stop sending at different times thereby moving the system from undersubscribed through oversubscription and back to undersubscribed before a barrage of web-like flows start. The 500 web-like flows entering R_1 are evenly distributed across twenty-five nodes, and each of these nodes is connected to R_1 again with propagation delays evenly spread between 1 and 100ms. The other twenty-five nodes generating web-like bursty traffic to R_2 are similarly connected. We use Barford and Crovella's web model [2]. In this model, each user downloads a set of files representing a web page and then sleeps for a period of time. The requested file size and sleep times obey heavy-tailed distributions and thus the resulting traffic is quite bursty. In Figure 6(a), we show the web users entering at R1 and R2 while in fact this means that the bulk of the traffic they generate enters R1 and R2. Because clients download more data than they upload, this implies that the web servers are placed at R1 and R2 while their respective clients are attached to R2 and R3.

To determine the target rate allocation when there is no oversubscription, we subtract the EMRs from the capacities in each bottleneck and then compute the weighted proportional fair share. Let K denote the number of bottlenecks. Let M denote the sum of the number of control loops passing through all three bottlenecks and the number of cross-flows entering each bottleneck. Thus, the target rate allocation x^*_{ij} for the flow (i,j) is

$$x^*_{ij} = \begin{cases} \frac{w_{ij}}{W}C + x_{ije} & if \quad i = 0 \\ \frac{w_{ij}}{W_i}(1 - \frac{W_0}{W})C & if \quad i \neq 0 \end{cases} \tag{28}$$

where $C = 100 - \sum_{0\leq j<M} x_{0je}$, $W_i = \sum_{0\leq j<M} w_{ij}$ and $W = \sum_{0\leq i<K} W_i$.

If a control loop with an EMR x_{0je} incurs its accumulation limit, we simply set its EMR to zero and replace its weight with the control loop's accumulation limit (to compute its degraded service rate). We describe the following cases.

1) No Oversubscription: As specified in Table I, long flows labelled $(0,0)$ and $(0,3)$ as well as all cross flows that traverse a single bottleneck start at a uniformly distributed random time in 0∼5s, denoted as $U[0,5]$s. The sum of the EMRs for $(0,0)$ and $(0,3)$ is 40Mbps, well below the bottleneck capacity. Neither flow needs to incur an accumulation greater than its accumulation limit to achieve its EMR. Thus, as shown in Figure 6(b), both flow $(0,0)$ and $(0,3)$'s EMRs are satisfied and they obtain their respective target rates as determined by Equation (28).

2) Accumulation-limited: At 25s in the simulation, flow $(0,1)$ begins transmitting and steers toward an EMR of 35Mbps. The sum of the active EMRs 75Mbps is still less than the capacity, but for flow $(0,3)$ to achieve its expected rate would require $a_{03} > A_{03}$. Therefore, flow $(0,3)$ becomes bounded by its accumulation limit shown in Figure 6(d) and fails to obtain its EMR. Even though x_{00e} and x_{01e} are larger than x_{03e}, they are satisfied because we have a policy of giving them larger accumulation limits.

3) Oversubscription: At 50s, flow $(0,2)$ begins transmitting, resulting in blatant oversubscription. Because all of the EMRs are themselves less than the capacity, we would intuitively desire to have a subset of the EMRs satisfied. Unfortunately, flows $(0,0)$, $(0,1)$ and $(0,2)$ all have similar accumulation limits thereby forcing all to a weighted proportionally fair share satisfying none of the EMRs. But none of the flows without an expected rate are starved.

At 75s, flows $(0,1)$ and $(0,2)$ stop sending thereby allowing the system to return to an equilibrium that satisfies all expected rates for active flows. Throughout the simulation, x_{03} changes slowly compared to x_{00},

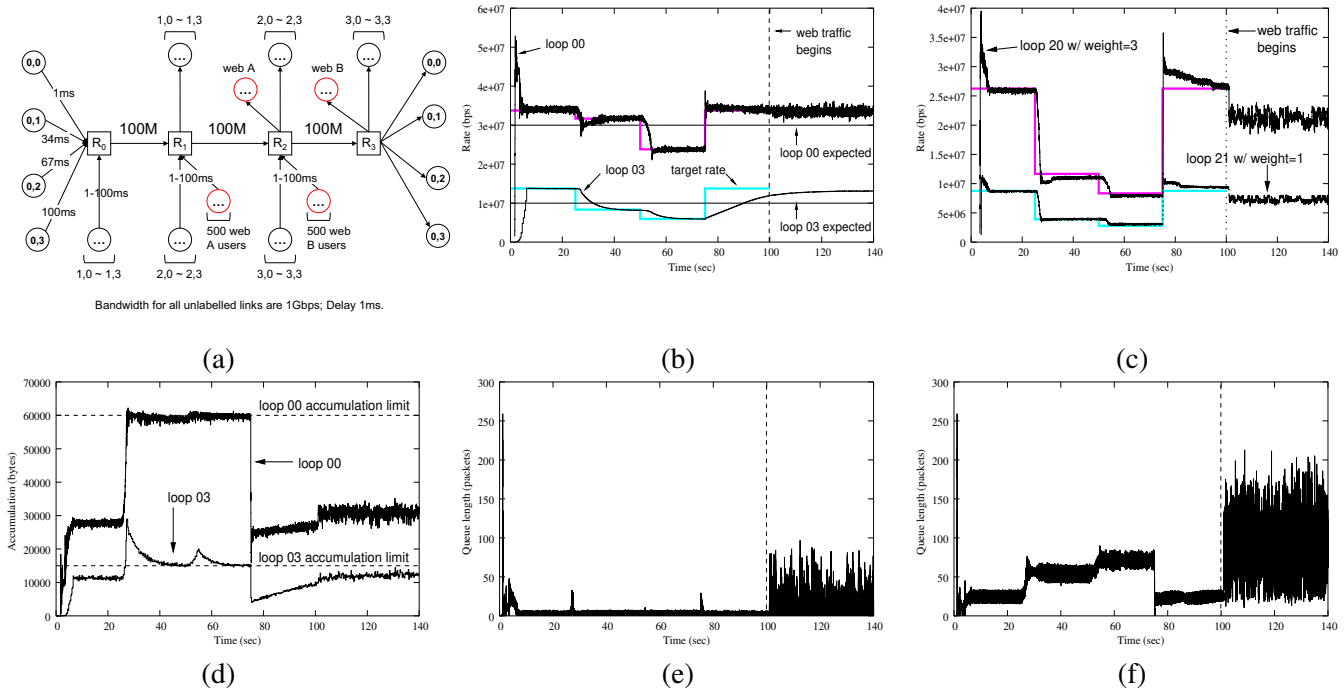


Fig. 6. Simulation of all services co-exist in a complex network of three bottlenecks: (a) A parking lot topology of 3 bottlenecks shared by long and short flows, plus 1,000 web sessions; (b) EMR flow throughput: Flow (0,0) gets its expected 30Mbps bandwidth, except during 50∼75s when its accumulation limit of 60KB takes effect; Flow (0,3) gets its expected 10Mbps bandwidth, except during 25∼75s when its accumulation limit of 15KB takes effect; (c) WRS flow throughput: Flow (2,0) gets 3 times of bandwidth of flow (2,1); (d) Accumulation for flows with rate expectations. When the accumulation of an EMR flow touches its preset limit, the EMR degrades, as shown in (b); (e) AVD bottleneck R_1 queue length is very low, even after web traffic comes in; (f) Droptail bottleneck R_2 queue length is proportional to the number of active flows. It is made more bursty by web traffic during 100∼140s. We set κ=0.5, target accumulation a_{ij}=3 $*$ w_{ij}KB, AVD's damping factor α=0.1 and target utilization β=0.98.

shown in Figure 6(b). This can be attributed to the large difference in round trip propagation delays; Flow $(0,0)$ has 10ms while flow $(0,3)$ has 208ms. However, despite their difference in propagation delays these flows still converge to the appropriate target rate allocation throughout the simulation, or at least until web-like traffic begins at 100s, at which time the equilibrium is no longer well-defined.

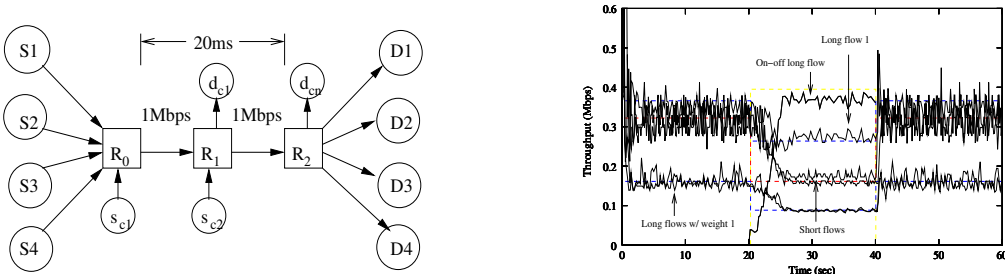
4) Web-like Traffic: At 100s, five hundred web users entering bottleneck R_1 and five hundred web users entering bottleneck R_2 introduce substantial variation in queue lengths, illustrated in Figures 6(e)-(f). We determined empirically an appropriate number of web users by turning off all sources except web users. We then tuned the number of web users to consume approximately 10% capacity. Of course, due to burstiness loads are sometimes much higher. The key result is that despite burstiness, each control loop hovers above its expected rate shown in right side of Figure 6(b).

Flow $(0,3)$ does not appear to adjust much in response to bursty web users. We attribute this to its very long propagation delay and therefore slower adaptation.

5) Coexisting Bottleneck Mechanisms: Droptail and AVD: As shown in the topology Figure 6(a), bottleneck R_1 uses AVD while others use droptail without AQM. Figures 6(e)-(f) readily demonstrate the benefits of AVD. The bottleneck R_1 experiences equilibrium queue lengths near zero independent of the changing rate allocations and, when web traffic starts, the queues still remain substantially lower than the bottleneck R_2.

6) Weighted Sharing: Flow $(2,0)$ sends with weight 3 throughout the experiment. For comparison we show its neighboring flow $(2,1)$ with weight 1. Because these flows traverse the same path, we expect $(2,0)$ to obtain roughly three times the throughput of $(2,1)$ regardless of the changing rate expectations or the presence of web-like flows. Figure 6(c) reveals this. Furthermore, it shows that as the load from EMR flows changes, each control loop steers toward the new rate allocation corresponding to proportional fairness for the capacity not allocated to satisfied EMRs.

This simulation demonstrates that all the proposed services can be provided in a dynamic environment with non-trivial bursty background traffic and the target rate allocations are well-defined and controllable via our choice of accumulation limits A_i even under oversubscription.



(a) (b)

Fig. 7. Implementation of all services co-exist in a network of two bottlenecks: (a) Topology; (b) Throughput vs. time.

B. Implementation Experiments

Besides ns-2 simulations, we also implement Monaco in Linux kernel v2.2.18 [29] based on the Click configurable router [17]. We did a set of experiments on the Utah Emulab [20]. We show one result here for a two bottleneck network shown in Figure 7(a) with 1Mbps link bandwidth and 20ms one-way delay. There are four long flows which pass all bottlenecks and two cross flows each using one bottleneck. Long flow 1 asks for an expected rate of 0.2Mbps. Long flow 2 requests a weighted service with weight 5. All other flows have weight 1. Long flow 2 is an on-off flow with a period of 20s. We did the experiment for 60s. As depicted in Figure 7(b), each flow gets its targeted rate.

VII. Summary

In this paper we propose to use an accumulation-based, closed-loop congestion control mechanism to provide service differentiation, based on our prior work of using accumulation, buffered packets of a flow inside network routers as a measure to detect and control network congestion. The key idea is to map service differentiation as accumulation allocation. We design two concrete services: EMR and WRS. Analytically as well as experimentally, we show that accumulation can be appropriately manipulated to provide each specific service. We use a set of ns-2 simulations to evaluate the service performance under different topologies and conditions. We demonstrate that both services can be provided in a network with dynamic demands, under the conditions of oversubscription and buffer limits. We implement the scheme in Linux kernel based on the Click router and validate the simulation results using the Utah Emulab and an internal testbed.

This paper focuses on the data-plane building blocks for service provisioning. The related control plane functions and architectural issues, such as the mapping in an edge-to-edge manner to provide cross-ISP services, represent our ongoing research.

VIII. Acknowledgments

The authors thank Josh Hort, Kishore Ramachandran and Rahul Sachdev for their help.

References

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

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

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

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

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

[6] 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.

[7] D. Choe and S. Low. Stabilized Vegas. *To Appear in Proc. INFOCOM'03*, April 2003.

[8] D. Clark and W. Fang, Explicit Allocation of Best Effort Packet Delivery Service. *IEEE/ACM Trans. on Networking*, 6(4):362-373, August 1998.

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

[10] A. Demers, S. Keshav, and S. Shenker. Analysis and Similation of a Fair Queueing Algorithm. *Procedings of SIGCOMM'89*, September 1989.

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

[12] Frame Relay Forum. Http://http://www.frforum.com/.

[13] R. Guérin et al. Scalable QoS Provision Through Buffer Management. *Proc. SIGCOMM'98*, September 1998.

[14] D. Harrison. Edge-to-edge Control: A Congestion Avoidance and Service Differentiation Architecture. *RPI CS Ph.D. Thesis*, December 2001. Available at http://networks.ecse.rpi.edu/~harrisod/thesis.ps.gz.

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

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

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

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

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

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

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

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

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

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

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

[26] 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, June 1993.

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

[28] I. Stoica. Stateless Core: A Scalable Approach for Quality of Service in the Internet. *CMU CS Ph.D. Thesis*, December 2000.

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

[30] Z. Wang. User-Share Differentiation (USD): Scalable Bandwidth Allocation for Diffrentiated Services. *IETF Draft*, November 1997.

[31] Y. Xia, D. Harrison, S. Kalyanaraman, K. Ramachandran and A. Venkatesan. An Accumulation-based Congestion Control Model. *To Appear in IEEE International Conference on Communications*, Anchorage, Alaska, May 2003. Available at http://www.rpi.edu/~xiay/pub/acc.ps.gz.

APPENDIX

For the reviewers' convenience, we provide here the background theory for Sections IV and V. Please refer to [31] for a complete description.

Consider the network defined in Section IV-A. Let's firstly analyze from queuing system perspective [23]. After the system approaches a steady state, at any link l the queue length $q_l\ (=\sum_{i\in I_l} q_{il})$, or equivalently the queuing delay $t_{ql}\ (=q_l/c_l)$, could be non-zero *only* if the capacity c_l is fully utilized by the sharing flows of the aggregate rate $\sum_{i\in I_l} x_i$, where x_i is the rate of flow i. This suggests either $q_l = 0$ (i.e., $t_{ql} = 0$) or $\sum_{i\in I_l} x_i = c_l$. We use window-based flow control, in which a window w_i bits of flow i could be stored either in node buffers as accumulation $a_i\ (=\sum_{l\in L_i} q_{il})$ or on transmission links as $x_i \cdot rtt_{p\,i}$, where $rtt_{p\,i}$ is the round trip propagation delay of flow i. Observing that $w_i = x_i \cdot rtt_i$, we summarize to get the following:

Proposition 1: If we use accumulation a_i as a steering parameter to control flow i's congestion window size w_i, then at the steady state (achievable by the control algorithm in Section III-A.2) we have, $\forall i \in I, \forall l \in L$:
(a) $w_i = a_i + x_i \cdot rtt_{p\,i} \Rightarrow a_i = x_i \cdot \sum_{l\in L_i} t_{ql}$;
(b) $t_{ql} \cdot\ (c_l - \sum_{i\in I_l} x_i) = 0$;
(c) $\sum_{i\in I_l} x_i \le c_l$;
(d) $t_{ql} \ge 0$;
(e) $x_i > 0$.

Alternatively, network resource allocation can also be modelled as a nonlinear optimization problem [16] [21] [18]. 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 $\sum_{i\in I_l} x_i \le c_l$ at all links. Using Lagrange multiplier method, we construct a Lagrange function $L(\vec{x},\vec{p}) = \sum_{i\in I} U_i(x_i) + \sum_{l\in L} p_l \cdot\ (c_l - \sum_{i\in I_l} x_i)$. If $U_i(x_i)$ is defined as $U_i(x_i) = s_i\, ln x_i$, where $s_i > 0$ is a weight, then because of the strict concavity of the objective function constrained by a convex set, we apply the Karush-Kuhn-Tucker condition to obtain:

Proposition 2: The nonlinear programming problem

$$\begin{aligned} maximize \quad & \sum_{i\in I} s_i\, ln x_i \\ subject\ to \quad & \sum_{i\in I_l} x_i \le c_l,\ \forall l \in L \\ & x_i > 0,\ \forall i \in I \end{aligned} \tag{29}$$

has a unique global maximum and $\vec{x}$ is the maximum if and only if, $\forall i \in I, \forall l \in L$:
(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) $\sum_{i\in I_l} x_i \le c_l$;
(d) $p_l \ge 0$;
(e) $x_i > 0$.

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

$$U_i(x_i) = a_i\, ln x_i. \tag{30}$$

It turns out that the weight s_i is instantiated by accumulation a_i which has a clear physical meaning and could

be manipulated to provide a set of QoS.

Besides, the Lagrange multiplier p_l is a measure of congestion, or price explored in [21], 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. In a FIFO router, the Lagrange multiplier $t_{ql} = q_l/c_l$ is provided by a physical FIFO queuing process where c_l is fixed and we have no freedom to control q_l. Similar to AVQ [19], we can also provide the same Lagrange multiplier t_{ql} using an active queue management algorithm such that $t_{ql} = q_l'/c_l'$ and q_l' is bounded if we change the virtual capacity c_l' accordingly. In this case we call t_{ql} virtual delay.