A Closed-loop Scheme for Expected Minimum Rate and Weighted Rate Services

David Harrison[†], Yong Xia, Shivkumar Kalyanaraman, Arvind Venkatesan[†]

CS[†] and ECSE Departments

Rensselaer Polytechnic Institute, Troy, NY 12180, USA

Contact email: 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 [28] 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 has a significantly larger range than achieved by the loss-based approaches in [8] [21]. 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 optimization framework [14]. This scheme does not require admission control. Instead, during oversubscription it degrades to a well-defined, policy-controlled bandwidth allocation. We use ns-2 simulations and Linux kernel implementation experiments to demonstrate that the service performance matches theoretic prediction. Our scheme does not require Active Queue Management (AQM) at bottlenecks. However, with AQM, we achieve near zero queue with high utilization.

I. Introduction

As the Internet evolves, its best-effort service model must be augmented to support more application requirements. This paper proposes the use of 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. Services based on the expected minimum rate building block are 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) [10]. EMR services are also

[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.

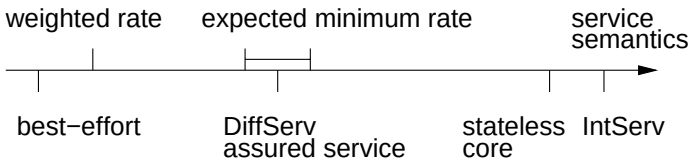


Fig. 1. Service spectrum

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 [23] [11] [27]. DiffServ relies on proper closed-loop congestion control (e.g. TCP [13]) to converge on a target bandwidth allocation. The overall performance is determined by the interaction of these two aspects. With the EMR building block, we explore how to further remove complexity from the network while expanding the range of services enabled. For example, unlike DiffServ, EMR has a well-defined equilibrium under oversubscription.

The weighted rate service building block provides a weighted proportionally fair allocation of the network capacity. Mechanisms realizing weighted proportional fairness are found throughout the nonlinear optimization-based congestion control literature [14] [19] [16] [20]. We demonstrate the wide achievable range of weights comparing to the loss-based closed-loop approaches [8] [21].

We develop concrete service building blocks based upon our prior work on a family of Accumulation-based Congestion Control (ACC) schemes [28] that use accumulation, the number of buffered packets of a flow inside the network, as a congestion measure. An ACC control loop measures its accumulation and adjusts its sending window to keep the accumulation around a target value. In [28] we proved that ACC is globally stable, achieves weighted proportional fairness, and has 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 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 [6] [28] incurred on a virtual queue [17]. 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 [13], DiffServ [3], to rate/delay guaranteed service offered by IntServ [4], including frameworks such as Stateless Core (SCORE) [25]. 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 expected minimum rate and weighted rate building blocks using ns-2 [22] 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 functionality placement and multi-ISP service coordination, etc.

This paper is organized as follows. In Section II we describe one of the ACC schemes, Monaco, that serves as the basis on which we provide the expected minimum rate service in Section III and the weighted rate service in Section IV. A set of ns-2 simulation results are provided in Sections III-B and IV-A. We multiplex all the services in a complex network to show that they can co-exist in Section V. Section VI concludes this paper.

II. Background

In this section we describe briefly the Monaco scheme (see [28] for more details) based on which the expected minimum rate and weighted rate services are provided.

A. Accumulation

In the bit-by-bit fluid model, at any time t, we define a flow i's accumulation $a_i(t)$ as a time-shifted, distributed sum of its queued bits in all the J nodes along its path, namely,

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

where $q_{ij}(t - \sum_{k=j}^{J-1} d_k)$ is flow i's bits queued in the jth node at time $t - \sum_{k=j}^{J-1} d_k$, and d_k is the propagation delay from up-stream node k to down-stream node k+1. Note this definition includes only those bits backlogged inside the buffers of all the nodes on the path, not those stored on transmission links.

Based on the measurement and control of accumulation, we have shown that a family of congestion control schemes can be derived. One example is TCP Vegas [5], 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. A variety of accumulation estimators are available that mitigate Vegas' sensitivities and may be used

with the service building blocks in this paper. In particular, we designed a receiver-based, out-of-band scheme that ensures robust accumulation estimation [28].

B. Monaco Scheme

Monaco employs a window-based, proportional control policy. It tries to maintain a constant target accumulation a_i for each flow i according to

$$cwnd_i(n+1) = cwnd_i(n) - \kappa \cdot (\hat{a}_i - a_i) \tag{2}$$

where $\hat{a}_i$ is the accumulation estimate, κ is set to 0.5, and $cwnd(n)$ is the congestion window value at a control period n. In addition we 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 packets lost.

This control policy fits within a set of functions described by the general Accumulation-based Congestion Control (ACC) model, which we introduced in [28]. We therein proved that functions meeting the ACC model are globally stable, realize weighted proportional fairness, and the equilibrium bandwidth share is determined by the weight which turns out to be the target accumulation a_i. It is this weight a_i that enables the capability of service differentiation. In principle, since the equilibrium is decided by a_i, we can accordingly adjust a_i to steer the equilibrium in order to achieve a specific service, e.g., expected or weighted rate allocations in Sections III and IV.

The ACC model also gives us the freedom to choose a control policy within certain constraints. As argued in Section IV-B, this decoupling of setting the target steady state allocation (fairness) from designing the control policy (stability and dynamics) allows us to avoid the major pitfalls in the loss-based service differentiation approaches.

III. Expected Minimum Rate Service

In this section we demonstrate an expected minimum rate service building block. We aim to provide any flow *a contracted (or expected minimum) bandwidth plus a proportional share of remaining capacity*. The expected minimum rate is achieved by keeping an appropriate amount of accumulation 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 II-B, 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 expected minimum rate x_{ie} and a proportional share x_{ip}^* of the remaining capacity[2]:

$$x_i^* = x_{ie} + x_{ip}^*. \tag{3}$$

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.,

$$a_i = a_{ie} + a_{ip}. \tag{4}$$

According to Little's law, we have

$$a_i = x_i^* \cdot t_{iq}, \tag{5}$$

$$a_{ie} = x_{ie} \cdot t_{iq}, \tag{6}$$

$$a_{ip} = x_{ip}^* \cdot t_{iq} \tag{7}$$

where t_{iq} is the steady state queueing delay experienced by flow i in its forward path. Equations (4)-(7) lead to

$$a_{ip} = (1 - \frac{x_{ie}}{x_i^*}) \cdot a_i. \tag{8}$$

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 (6). To avoid setting a varying parameter a_{ie}, or equivalently a_i in Equation (5), we use Equation (8) to calculate a_{ip} which automatically accounts for a_{ie}. Then we just need to keep a_{ip} constant.

Assume enough buffers are provided in all congested routers and there is no oversubscription. Firstly we consider the total rate x_i^* achieved with corresponding accumulation a_i. As proved in [28][3], the flow rate x_i^* is the unique

[2]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.
[3]Here we don't repeat the proof procedure, which is available at http://www.rpi.edu/~xiay/pub/acc.ps.gz.

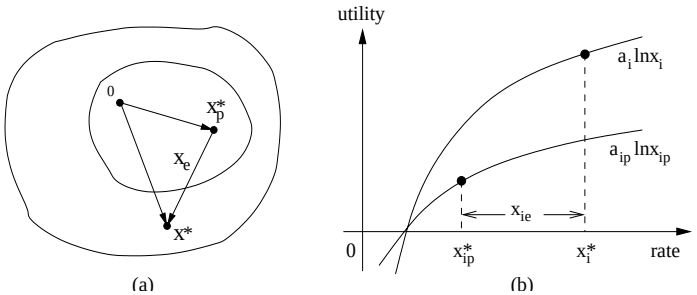


Fig. 2. Two simultaneous nonlinear optimization problems: (a) Convex constraint sets showing $\vec{x^*} = \vec{x_e} + \vec{x_p^*}$: inner one for x_{ip} while outer one for x_i; (b) Utility functions of x_i and x_{ip}.

maximum of the following nonlinear optimization problem related to a_i:

$$maximize \quad \sum_{i \in I} a_i \, ln x_i \tag{9}$$

$$subject\ to \quad \sum_{i \in I_l} x_i \leq c_l, \ \forall l \in L \tag{10}$$
$$x_i > 0, \ \forall i \in I$$

meaning that x_i^* achieves weighted proportional fairness:

$$\sum_{i \in I} a_i \cdot \frac{x_i - x_i^*}{x_i^*} \leq 0 \tag{11}$$

where x_i is any feasible rate satisfying constraint (10).

Beyond the expected rate x_{ie}, we secondly 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}:

$$maximize \quad \sum_{i \in I} a_{ip} \, ln x_{ip} \tag{12}$$

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

meaning x_{ip}^* achieves similar proportional fairness:

$$\sum_{i \in I} a_{ip} \cdot \frac{x_{ip} - x_{ip}^*}{x_{ip}^*} \leq 0 \tag{14}$$

where x_{ip} is any feasible rate satisfying constraint (13).

So a system providing an expected minimum rate 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 2. The reader is referred to Section 5.3 of [12] 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 the expected minimum rate:

- Oversubscription: Without admission control, the constraint (13) might be always invalid. Or it is possible that

$$\sum_{i \in I_l} x_{ie} \geq c_l, \ \exists \, l \in L; \tag{15}$$

- 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,

$$a_i = \sum_{L \in L_i} q_{il} \leq A_i, \ \forall i \in I; \tag{16}$$

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

$$q_l = \sum_{i \in I_l} q_{il} \leq Q_l, \ \forall l \in L. \tag{17}$$

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

Firstly, we argue that the constraints (15) and (17) can be translated equivalently into the form of (16). For (17), it is obvious because both (16) and (17) are constraints on q_{il}. For (15), 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 (16) will be effective. So these three constraints can be summarized into one form of (16) with an equivalent accumulation limit A_i^{eq}. To take into account all the above boundary conditions (15)$\sim$(17), we define a new optimization problem similar to (9) but with changed coefficients a_i' and thus different, even still logarithmal, utility functions:

$$\begin{aligned} maximize \quad & \sum_{i \in I} a_i' \, ln x_i' \\ subject \ to \quad & \sum_{i \in I_l} x_i' \leq c_l, \ \forall l \in L \\ & x_i' > 0, \ \forall i \in I \end{aligned} \tag{18}$$

where

$$a_i' = min(a_i, A_i^{eq}), \ \forall i \in I. \tag{19}$$

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

$$\sum_{i \in I} a_i' \cdot \frac{x_i' - x_i^{'*}}{x_i^{'*}} \leq 0 \tag{20}$$

where x_i' is any feasible rate.

Consequently, when there is either oversubscription, accumulation limit, or underprovisioned buffers, these conditions affect the realization of (9) and (12), i.e., the expected rate can not be provisioned. A new optimization problem (18) is automatically defined by adding these constraints (15)∼(17) into (9), 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 (19). As a result, the expected service gracefully degrades into the weighted service discussed in Section IV. Further, since we have the freedom to set the accumulation limit A_i for each flow i in (16), 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 steady state queue length bounded? We have adopted the AVQ algorithm [17] which emulates an adaptively changing link capacity such that the steady state queue length is sufficiently small. We compute a virtual queueing delay [6] 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[4]. 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 V and [28]). 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 expected minimum rate service. It includes seven steps.

In Step 1, we compute the departure rate x as the 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 expected minimum rate according to Equation (8). When a control loop is ramping up or during oversubscription, the departure rate x may be less than the expected minimum rate 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

[4]This accumulation estimator (see Section II-A) 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].

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 expected minimum rate
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 $(a > A \mid\mid a_p > \varepsilon)$ { $ssthresh = cwnd$; }
(6) else {
  (6.1) if $(pwnd + mtu >= ssthresh)$
    $ssthresh = cwnd$;
  (6.2) $cwnd = min(pwnd * 2.0, ssthresh)$; }
(7) rate_limit = $cwnd * 8.0/srtt$;

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 (2) 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 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. Simulations

We evaluate the performance of the expected minimum rate service building block using ns-2 simulations on a single bottleneck. Each flow has a different propagation delay. In Section V 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 the Monaco scheme. We use simulation because this allows us to temporarily set aside the practical limitation of finite buffer

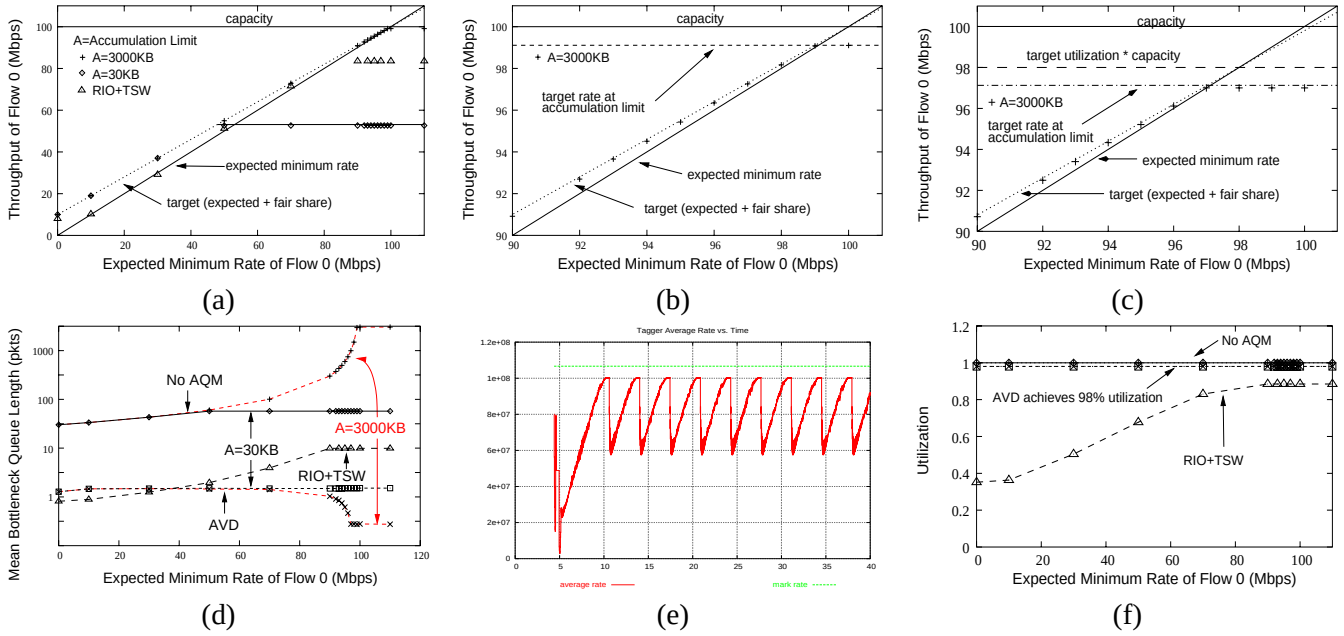


Fig. 3. Expected minimum rate in a single bottleneck: (a) Monaco throughput x_0 vs. expected minimum rate x_{0e} with non-AQM droptail bottleneck and RIO throughput vs. DiffServ bandwidth allocation; (b) The former graph zoomed into 90-100Mbps; (c) Monaco throughput x_0 vs. expected minimum rate x_{0e} with AVD bottleneck, zoomed into 90-100Mbps; (d) Steady state queue length vs. expected minimum rate 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 [7].

sizes, so that we can study the range of services in the absence of packet loss. Flow 0 has an expected minimum rate x_{1e} while other nine flows request a proportional rate service (i.e., with an expected minimum rate 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.

In the first set of simulations we evaluate the range of satisfiable expected minimum rates 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 3(a) and zoomed in 3(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 expected minimum rate is $\frac{30\times100}{30+9\times3} = 52.6$Mbps, demonstrating that the unsatisfied expected minimum rate due to the constraint (16) degrades to a weighted rate (see Section IV).

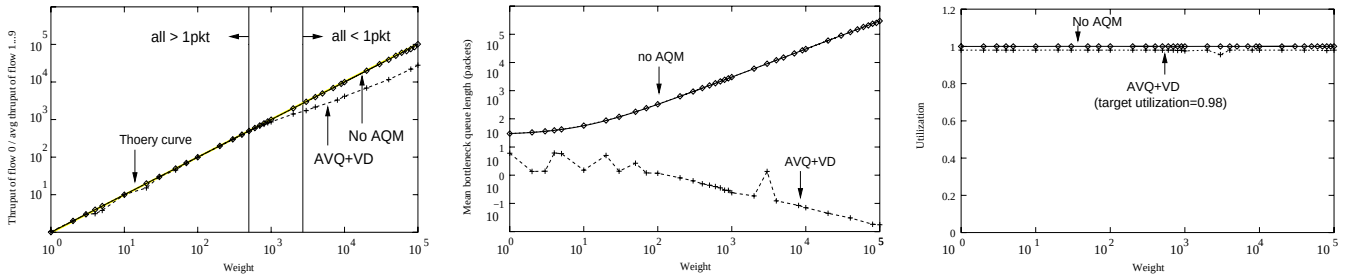
The upper part of Figure 3(d) shows that the queue length grows dramatically (note the logarithmic scale of the vertical axis) as the expected minimum rate x_{1e} increases. This is most apparently shown by the case when $A = 3000$KB. However in all cases, the queue growth flattens when the accumulation limit is reached. The rapid queue growth as the expected minimum rate 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 of the expected minimum rates with AVD bottleneck. The range of achieved expected minimum rates is similar to the case without AQM, as shown in Figure 3(c) for the expected rate of 90-100Mbps. However, AVD keeps the queue length at near zero as depicted in the lower part of Figure 3(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 because 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 3(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 the expected minimum rate 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 [7]. They propose that each user has a "profile" specifying how much capacity should be allocated to TCP connections during congestion. They place a component 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 drops packets randomly during congestion with preference toward keeping packets that are marked in-profile. To perform marking he proposes the Time Sliding Window (TSW) tagger and to perform dropping he proposes RIO. The TSW tagger 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 [9] algorithms in parallel. One of the RED algorithms randomly drops "in" packets with increasing probability on the number of "in" packets in the queue and the other randomly drops "out" packets with increasing probability on the number of "in+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 [7]. Additionally our RIO algorithm has Explicit Congestion Notification (ECN) [24] turned on. Thus packets are marked for congestion (different than the in/out mark) when RIO would otherwise have dropped.

In Figure 3(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 additive-increase-multiplicative-decrease (AIMD) algorithm and thus affects all profile marking, ECN marking, and AQM algorithms. Since the



(a) (b) (c)

Fig. 4. Weighted rate 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.

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 3(d)), limits on achievable allocations (as shown for bandwidth allocations above 80Mbps in Figure 3(e)), or low utilization (as shown in Figure 3(f) for bit rates below 80Mbps). This *fundamental limitation* of TCP+RIO makes it impossible to simultaneously achieve high utilization, low queue length, and a wide disparity in the allocated rates. By avoiding AIMD, Monaco with AVD simultaneously achieves all three.

IV. Weighted Rate Service

As stated in Section II-B, our control policy achieves weighted proportional fairness. In this section we show the range of weights achievable with droptail or AVD bottleneck using the control policy in Equation (2). The achieved weighted rate service range significantly outperforms the studied loss-based approaches in [8] [21] by *three orders of magnitude*.

A. Simulations

We evaluate the performance of the weighted rate 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 V.

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 three orders of magnitude larger than the weight (10∼100) achieved with TCP [8] [21]. We provide theoretic reasoning for this huge difference in the next subsection. As shown by the upper curve in Figure 4(a), for the wide range of weight variation, accurate weighted sharing is

achieved. This comes with a cost. The upper curve in Figure 4(b) shows that the steady state queue length at the bottleneck increases linearly with weight. The curvature in the mean queue length in Figure 4(b) is due to a y-offset for a weight 1 equal to the sum of the target accumulations (30KB). As with the expected minimum rate 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 4(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 expected minimum rates increase as we already discussed in Section III-B: control loops that send rarely also rarely perturb the queue.

As shown in Figure 4(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 does not allow the window size of the control loop with the heavy weight to grow as would occur with growing physical queue length, but instead forces the window size of each control loop with weight 1 to shrink. 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 4(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 4(c), without AQM, the utilization is 100%. With AQM the utilization settles around 98%, which is our target utilization for AVD.

B. Accumulation-based vs. Loss-based Approaches

Related research, such as [8] [21], 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 its steady state objective and at the same time maintain good dynamic performance.

Crowcroft et al. implement MulTCP which makes a TCP connection behave as w TCP Reno 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 $\frac{w-0.5}{w}$ [8]. Nandagopal et al. provide a systematic analysis on how to adjust TCP Reno congestion control increase/decrease parameters to achieve the weighted service [21]. 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 [21] 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 (2). The control algorithm's dynamic behavior can be designed separately as long as it fits into the general ACC model [28]. 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 (17), bottleneck buffer size limits the range of the weighted service in practice. But this limitation is very different from that of [8] [21] where it is resulted from the coupling of steady state objective and dynamic performance.

V. Service Multiplexing

In this section we provide ns-2 simulation and Linux kernel implementation experiment results to demonstrate that the expected minimum rate and weighted rate services can *co-exist dynamically* with bursty 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 5(a). We choose a topology with all equal capacities and put all control loops with expected minimum rates 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,

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	

each of the long flows has an expected minimum rate, 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 five hundred 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.

To determine the target rate allocation when there is no oversubscription, we subtract the expected minimum rates 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

$$x^*_{ij} = \begin{cases} \frac{w_{ij}}{W} C + x_{ije} & if \quad i = 0 \\ \\ \frac{w_{ij}}{W_i}(1 - \frac{W_0}{W})C & if \quad i \neq 0 \end{cases} \tag{21}$$

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 expected minimum rate x_{0je} incurs its accumulation limit, we simply set its expected minimum rate 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 expected minimum rates 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 expected minimum rate. Thus, as shown in Figure 5(b), both flow $(0,0)$ and $(0,3)$'s expected minimum rates are satisfied and they obtain their respective target rates as determined by Equation (21).

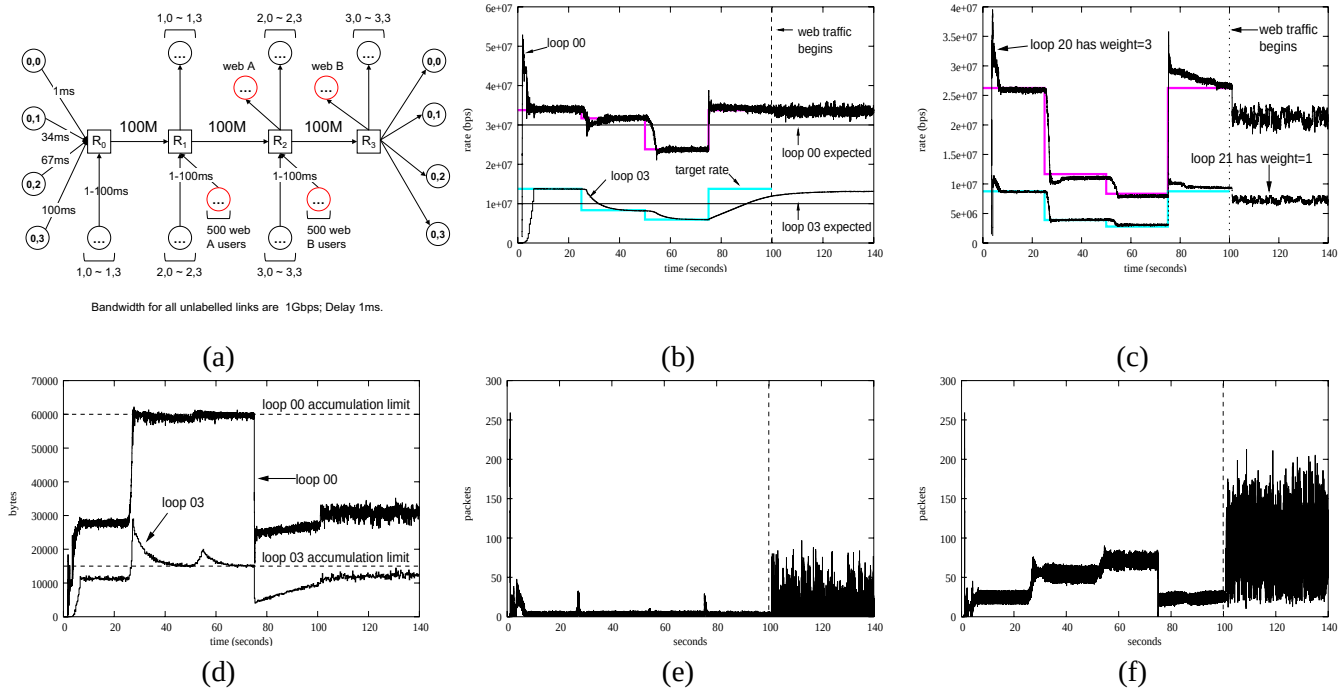
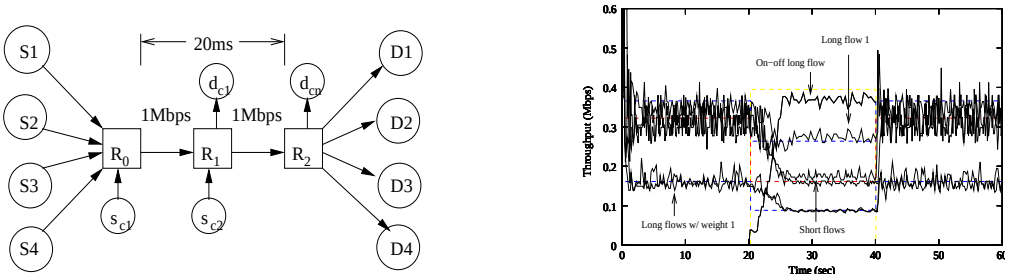


Fig. 5. Simulation of all services co-exist in a complex network of three bottlenecks: (a) Topology; (b) Throughput vs. time with rate expectations; (c) Throughput vs. time with weighted shares; (d) Accumulation vs. time for flows with rate expectations; (e) AVD bottleneck R_1 queue length vs. time; (f) Droptail bottleneck R_2 queue length vs. time. In these simulations, we set κ=0.5, target accumulation a_{ij}=w_{ij}*3KB, AVD's damping factor α=0.1 and target utilization β=0.98.

2) Accumulation-limited: At 25s in the simulation, flow $(0,1)$ begins transmitting and steers toward an expected minimum rate of 35Mbps. The sum of the active expected minimum rates 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 5(d) and fails to obtain its expected minimum rate. 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 expected minimum rates are themselves less than the capacity, we would intuitively desire to have a subset of the expected minimum rates 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 expected minimum rates. 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}, shown in Figure 5(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.



(a) (b)

Fig. 6. Implementation of all services co-exist in a network of 2 bottlenecks: (a) Topology; (b) Throughput vs. time.

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 5(e)-(f). We use the web models presented in [2]. 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 5(b).

Flow $(0, 3)$ does not appear to adjust much in response to bursty web flows. 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 5(a), bottleneck R_1 uses AVD while others use droptail without AQM. Figures 5(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 5(c) reveals this. Furthermore, it shows that as the load from expected minimum rates changes, each control loop steers toward the new rate allocation corresponding to proportional fairness for the capacity not allocated to satisfied expected minimum rates.

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.

B. Implementation Experiments

Besides ns-2 simulations, we also implement Monaco in Linux kernel v2.2.18 [26] based on the Click configurable router [15]. We did a set of implementation experiments based on our Monaco implementation using the Utah

Emulab [18]. We show one result here for a two bottleneck network shown in Figure 6(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 6(b), each flow gets its targeted rate.

VI. Summary

In this paper we propose to use a closed-loop congestion control mechanism to provide quality of services, based on our prior work of an accumulation-based congestion control model which uses accumulation, buffered packets of a flow inside network routers as a measure to detect and control network congestion. We design two concrete services: the expected minimum rate and weighted rate allocations. 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 the expected minimum rate and weighted rate 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. The reader is referred to [12] for more details.

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.

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

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

[8] 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.

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

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

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

[12] 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.

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

[14] 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.

[15] 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.

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

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

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

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

[20] L. Massoulie and J. Roberts. Bandwidth Sharing: Objectives and Algorithms. *Proc. INFOCOM'99*, March 1999.

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

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

[23] 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.

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

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

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

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

[28] Y. Xia, D. Harrison, S. Kalyanaraman, K. Ramachandran and A. Venkatesan. Accumulation-based Congestion Control. *Submitted to IEEE/ACM Trans. on Networking*, Janurary 2003. Available at http://www.rpi.edu/~xiay/pub/acc.ps.gz.