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

David Harrison†, Yong Xia, Shiv 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, we consider *closed-loop* techniques to achieve a range of service differentiation capabilities. Our key contribution is the use of Accumulation-based Congestion Control (ACC) 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. A user with a minimum rate expectation can be interpreted as having a non-concave user utility function. Unfortunately in the context of nonlinear optimization, non-concave utility functions are analytically difficult and often result in multiple optimal solutions. Instead of attempting optimization with non-concave objective functions, we demonstrate a meaningful notion of an expected minimum rate by imposing additional constraints on Kelly's convex optimization formulated in [14]. Unlike the constraint which simply requires all user allocations to be larger than their respective expected minima, our constraint does not require admission control. The resulting scheme is distributed, requires each control loop to act only on local knowledge and still allows policy-based control over how capacity is allocated during oversubscription. We use ns-2 simulations and Linux implementation experiments to demonstrate that the service performance matches theoretical results. 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 to a telecommunication infrastructure, its best-effort service model must be augmented to support more application requirements, such as service guarantees and service differentiation. This paper proposes the use of closed-loop mechanisms as a data-plane building block to provide expected minimum rate and weighted rate services. Expected minimum rate refers to a service that offers a minimum contracted rate assurance plus a proportional 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]. Expected minimum rate services are also similar to the DiffServ assured service [4], though the latter has multiple classes and drop precedences within each class. Both the DiffServ assured service and Frame Relay CIR/PIR service are realized using open-loop building blocks [11] [29]. The weighted rate service building block provides weighted proportional fairness. Mechanisms realizing weighted proportional fairness are found throughout the nonlinear optimization-based congestion control literature [14] [19] [16] [21]. We demonstrate the huge achievable range of weights and interoperation between our specific expected minimum rate and weighted rate building blocks.

We demonstrate that allocating minimum rate expectations can be practically interpreted as translating a concave objective by the rate expectations and applying additional convex constraints. In traditional QoS, expected minimum rate allocations are accomplished using admission control. In our framework, the closed-loop mechanisms operate in a fully distributed manner without admission control, and require each control loop to act only on local knowledge. The mechanisms allow policy-based control over capacity allocation during oversubscription by setting parameters on our introduced convex constraints.

We develop concrete service building blocks based upon our prior work on a family of Accumulation-based Congestion Control (ACC) schemes [30] that use the number of buffered packets of a flow inside the network as a congestion measure. We use the term *accumulation* to denote per-flow backlog in the path to differentiate accumulation from queue length or other notions of backlog. We argue that congestion control arising from this congestion measure can be superior to loss-based closed-loop approaches [9] [24] for service differentiation purposes.

This is because ACC schemes largely decouple the equilibrium rate allocation from the dynamics of their control algorithms, and thus do not require the system to become more aggressive in terms of the size of rate or window increase/decrease steps in order to obtain differentiation. 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 introducing AQM that communicates the *virtual queueing delay* incurred on a virtual queue [17][26].

We evaluate the expected minimum rate and weighted building blocks using ns-2 [25] simulations and Linux kernel v2.2.18 implementation experiments. Several fundamental issues such as graceful service degradation (like oversubscription, limits on accumulation), and tradeoffs (like FIFO 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. However, we remark that the abstract model developed in this paper offers the attractive potential of *overlaying* QoS capabilities over an existing network of logical FIFO queues from network edges.

Figure 1 illustrates a *qualitative* spectrum of packet-switched QoS capabilities ranging from best-effort service [13] to rate/delay guaranteed service offered by IntServ [6], including frameworks such as Stateless Core (SCORE) [27] and DiffServ [4]. Services on the right hand side of the diagram are more complex, offered at finer granularity, and have increasing implementation complexity. The left side of the spectrum implies less quantitative services, the use of FIFO queuing, closed-loop congestion control and potentially AQM schemes. Our framework would be in the middle of the spectrum emulating a subset of the DiffServ capabilities and extending the realm of services with closed-loop control, FIFO queueing, and optional AQM. Unlike IntServ, SCORE and DiffServ that offer per-packet assurances, our service semantics are meaningful only in the *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, and without the need for admission control.

This paper is organized as follows. In Section II we describe the control policy that serves as the base on which we provide an expected minimum rate service in Section IV and a weighted rate service in Section V. We also briefly discuss a key issue regarding the ACC model in general, i.e., scalability of buffer requirement and adopt

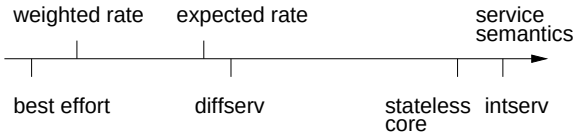


Fig. 1. Service spectrum

a virtual queuing mechanism as a solution in Section IV-A. We evaluate our proposal using a set of ns-2 simulations in Sections IV-B and V-A. We multiplex all the services in a complex network to show they can co-exist together in Section VI via ns-2 simulation and Linux OS kernel v2.2.18 implementation experiments. Section VII concludes this paper.

II. Control Algorithm

Define flow i's accumulation as the sum of the queued bits in all nodes along its path, i.e.,

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

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 node k to node $k+1$. Note it includes only those bits backlogged inside node buffers, not those stored on transmission links. We consider window-based congestion control in which each control loops tries to maintain a constant target accumulation a_i^* for each flow i according to

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

where $w_i(t)$, a_i^* and $a_i(t)$ are respectively the congestion window size, target accumulation, and instantaneous accumulation of flow i. In addition we bound the increase to within 1 MTU per round-trip time.

This control policy fits within a family of functions described by the general Accumulation-based Congestion Control (ACC) model, which we introduced in [30]. We therein proved that functions meeting the ACC model are globally stable, realize weighted proportional fairness (with weight a_i^*), and the equilibrium share is determined by the target accumulation. This allows us the freedom to choose a control policy within certain constraints (see [30]). We therefore choose a control policy that increases no more quickly than TCP Reno or Vegas (i.e., linearly), and backs off more quickly than Vegas in response to extreme queue lengths. As argued in Section V-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 loss-based service differentiation.

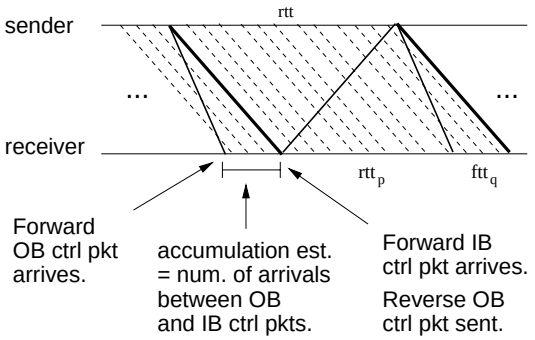


Fig. 2. Monaco accumulation estimator

III. Accumulation Estimation

ACC control policies potentially interoperate with any unbiased accumulation estimator operating on RTT timescales including the accumulation estimator introduced by TCP Vegas or the Monaco accumulation estimator we introduced in [30]. TCP Vegas estimates accumulation as $send\ rate \times (rtt - rtt_p)$ where rtt_p refers to Round-Trip Propagation Delay (RTPD). This accumulation estimator is sensitive to both the estimation of RTPD and to reverse path congestion[23] [30] [5]. In particular changes in the RTPD can lead to dramatic underutilization or grossly unfair bandwidth allocations. The Monaco accumulation estimator eliminates these sensitivities in the following way.

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.

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.

Due to the problems with the Vegas accumulation estimator, we use the Monaco accumulation estimator in all simulations and experiments in this paper.

IV. Expected Minimum Rate Service Building Block

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 any uncontracted capacity.* The expected minimum rate is achieved by keeping an appropriate amount of accumulation for that flow. We mean "expected" in the sense that we can obtain a high probability of meeting the contract under the assumption that the bandwidth allocated for the expected rate service is only a part of the total capacity. We show theoretic result for a general network topology based on the ACC model and evaluate it using a set of ns-2 [25] simulations based on Monaco.

A. Theory

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

As we discuss in Section II, 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 steady state accumulation.

Suppose we provide flow i a bandwidth x_i^* which is a sum of an contracted rate x_{ie} (which we call expected minimum rate hereafter) and a proportional share x_{ip}^* of the remaining network capacity [1]:

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

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 the forward path.

[1] 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.

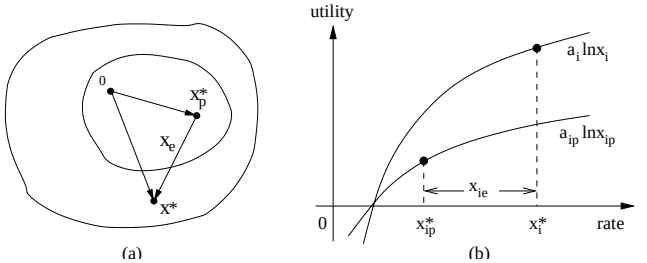


Fig. 3. Two simultaneous nonlinear optimization problems: (a) Convex constraint sets showing $\vec{x^*} = \vec{x_e} + \vec{x_p^*}$: inner one for x_{ip} while outer one for x_i; (b) Utility functions of x_i and x_{ip}.

Based on the above equations, we obtain

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

where x_{ie} is the contracted rate known *a priori*, x_i^* can be measured at receiver side, and a_i is provided by the Monaco accumulation estimator shown in Figure 2.

Obviously, if we keep a_{ip} constant, then the proportional share x_{ip}^* is decided by network as shown by Equation (7). On the contrary, if we want to keep a constant expected minimum rate x_{ie}, then the corresponding a_{ie} should be set according to Equation (6). To avoid the estimation of a_{ie} (or equivalently t_{iq}) we use Equation (8) to estimate a_{ip} which automatically accounts for a_{ie}.

To make the above observations more clear, we provide below a nonlinear optimization analysis. Firstly we consider the total rate x_i^*. As shown in Appendix (see also [30]), we prove that the flow rate x_i^* is the unique maximum of the following nonlinear optimization problem:

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

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

meaning that it achieves a weighted proportional fairness:

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

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

Beyond the contracted rate x_{ie}, we secondly consider the rate x_{ip}^*. Similarly we can prove that the flow rate x_{ip}^* is the unique maximum of the sub-problem:

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

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

meaning it also achieves another similar fairness:

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

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 3. A more general analysis is provided in Section 5.3 of [12] by considering general utility functions.

There are three boundary conditions affecting if these two optimalities can actually be realized in a practical system:

- Oversubscription: Without admission control, the constraint (13) might be always invalid. Or we can not guarantee that

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

- 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

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

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

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

Apparently, if any of the boundary conditions is effective, we have more constraints in the optimization problems (9) and (12). Thus the (optimal) expected service might not be achieved. But since the boundary conditions are all convex, the feasible region remains convex, then the system still has a unique equilibrium. Therefore the closed-loop control can still be used to achieve a weighted proportional rate allocation which can be computed from the boundary conditions. We use simulation to illustrate this in the next subsection. Consequently the expected service gracefully degrades into the weighted service discussed in the Section V. Further, since we have the freedom to choose the accumulation limit A_i in (16), this parameter could be used for policy-based control.

Now let's look at the trade-off. Even we set an accumulation limit for each flow, the steady state queuing delay

or physical queue length might be large as the number of multiplexed 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 (VD) which is defined as the ratio of queue length divided by virtual capacity [26] and add it into the forward in-band control packet. A nice property of the virtual delay algorithm is that it is incrementally deployable since a mixed set of FIFO and AVQ+VD routers can work together (see Section VI). In such an environment the Monaco accumulation estimation will be $\hat{a}_M = \hat{a}_{FIFO} + x \cdot \hat{t}_{VD}$ where $\hat{a}_{FIFO}$ is accumulation in those FIFO routers measured between two control packets shown in Figure 2, x is the egress rate and $\hat{t}_{VD}$ is the sum of all virtual delays at those AVQ+VD routers.

To sum up, we use the following algorithm to provide the expected minimum rate service. It includes eight steps.

Algorithm 1 Expected Service Pseudo-code at Ingress

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

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

In Step 1 we read from the reverse control packet the total accumulation a measured by the Monaco estimator.

In Step 2, we compute the departure rate x as the bits transmitted since the last control packet divided by the smoothed rtt $srtt$.

In Step 3, 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 4, we force a $pwnd$ that is within 1 MTU of $cwnd$ to be equal to $cwnd$. Our algorithm stops sending when the packet at the head of the queue would cause $pwnd$ to exceed $cwnd$. Thus anything within 1 MTU of $cwnd$ should be counted as being equal to $cwnd$.

In Step 5 we set the congestion window according to the Monaco control policy defined in Equation (2).

In Steps 6–7, we determine whether step 5 was a decrease or an increase step. In Step 6, 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. In Step 7.1, we only allow $ssthresh$ to increase if the number of bytes sent in the prior RTT is within an MTU of $ssthresh$. It also ensures that a cwnd that decreases due to something other than congestion (e.g., decreasing user demand), that $ssthresh$ does not decrease, i.e., $ssthresh$ tracks the congestion level in the network rather than the user demand. Step 7.2 bounds slow start by $ssthresh$.

Step 8 sets the rate on the token bucket shaper.

B. Simulations

In this subsection we evaluate the expected minimum rate building block performance using ns-2 simulations on a single bottleneck with heterogeneous propagation delays. In Section VI we provide ns-2 simulation and Linux kernel v2.2.18 implementation experiment results for a more complex multiple bottleneck network.

We simulate a single 100Mbps bottleneck with 4ms propagation delay shared by 10 flows using the Monaco scheme. 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 loss. The topology is shown in Figure 4(a). Flow 0 has an expected minimum rate x_{1e} while other 9 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 a 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 15KB, 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. With $A = 3000$KB, we are able to allocate 99.1Mbps (i.e., $\frac{3000 \times 100}{3000 + 9 \times 3}$) of a 100M bottleneck to a single flow before the accumulation limit is reached. When

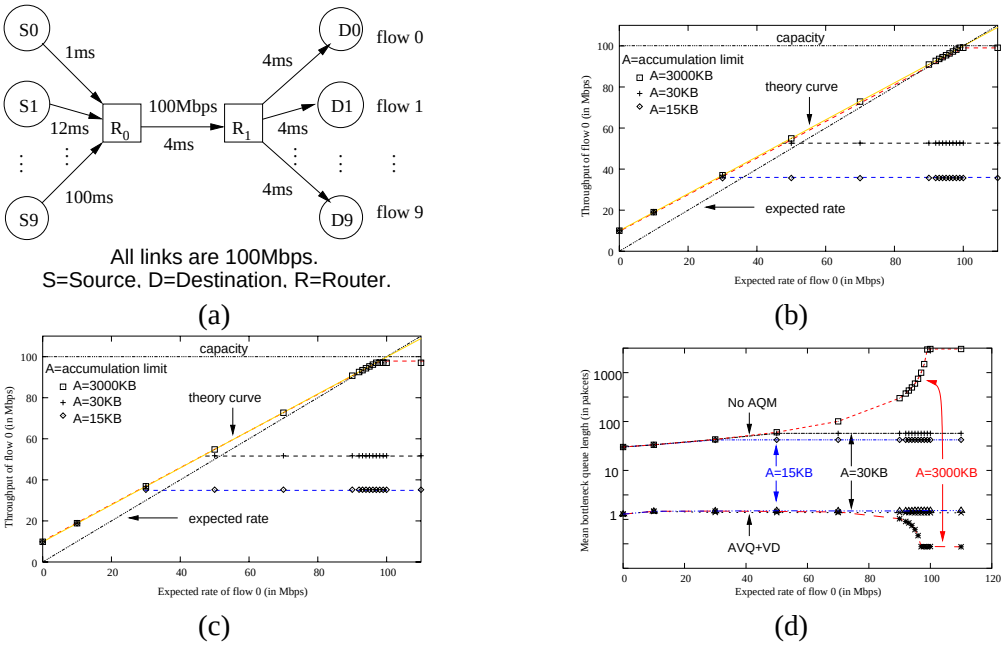


Fig. 4. Expected minimum rate block in a single bottleneck: (b) Topology; (b) Throughput x_1 vs. expected minimum rate x_{1e} without AQM; (c) Throughput x_1 vs. expected minimum rate x_{1e} with AVQ+VD at the bottleneck; (c) Steady state queue length vs. expected minimum rate x_{1e} without and with AQM at the bottleneck for flow 0's accumulation limit A set to 15KB, 30KB and 3000KB.

$A = 15$KB, the maximum satisfiable expected minimum rate, x_1, is $\frac{15\times100}{15+9\times3} = 35.7$Mbps, demonstrating that the expected minimum rate service degrades to a weighted rate (see the next section). Similarly, x_1 is limited to $\frac{30\times100}{30+9\times3} = 52.6$Mbps for $A = 30$KB.

The upper part of Figure 4(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 $A = 3000$KB case. 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 sizes in the network.

In the second set of simulation runs, we repeated the simulations above the expected minimum rates. We added AQM (more specifically AVQ+VD) to the bottleneck. The range of achieved expected minimum rates is similar to the case without AQM, as shown in Figure 4(c). However, AVQ+VD keeps the queue length at near zero in the equilibrium, because instead of incurring physical accumulation, each control 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.

Although not shown in the figures, the utilization for all cases without AQM was 100% in the steady state (the queue never drains completely), and when AQM was used, the utilization was always within half a percent of 98%, i.e., our target utilization for AVQ+VD.

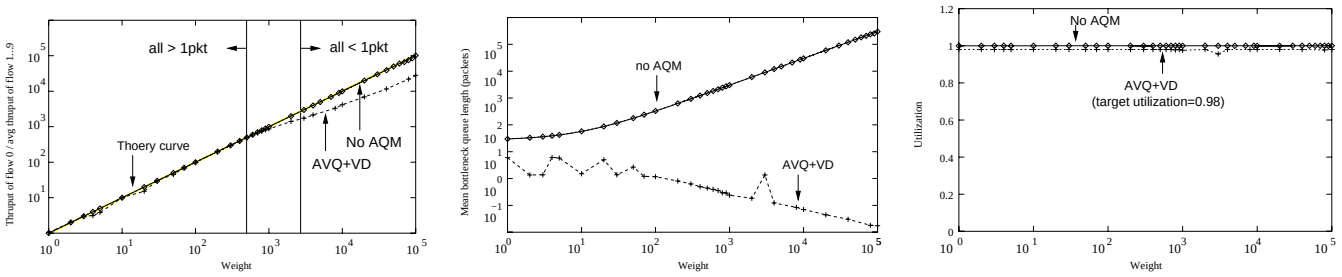
V. Weighted Rate Service

As stated in Section II, our control policy achieves weighted proportional fairness. In this section we demonstrate the range of weights achievable with and without AVQ-VD using the control policy in Equation (2) and show that the achieved weighted rate service range significantly outperforms the studied loss-based approaches [9] [24].

A. Simulations

In this subsection we evaluate the weighted rate service building block's performance using ns-2 simulations on a single bottleneck with large enough buffer size to avoid loss. For a network of multiple congested links shared by flows passing through different numbers of bottlenecks we provide ns-2 simulation and Linux kernel v2.2.18 implementation experiment results in Section VI.

We use the same single bottleneck network show in Figure 4(a) where a single 100Mbps 4ms bottleneck is shared by 10 flows with very heterogeneous RTTs. Flow 0 has varying weight w while other flows has unit weight (i.e, $a_0 = wa_1 = \cdots = wa_9$). We performed two sets of simulations to evaluate the weighted service range.



(a) (b) (c)

Fig. 5. Weighted service in a single bottleneck: (a) Relative throughput ratio vs. weight; (b) Bottleneck queue length vs. weight; (c) Bottleneck utilization.

In the first simulation 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 magnitudes* larger than the weight achieved with TCP [24]. We provide theoretic reasoning for this huge difference in the next subsection. As shown by the upper curve in Figure 5(a), for the enormously wide range of weight variation, accurate weighted sharing is achieved. This comes with a cost; 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 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 present with finite buffer sizes.

Again, with AVQ+VD we are able to break the coupling between the notion of accumulation and real queuing. Figure 5(b) demonstrates AQM achieving average queue sizes less than 1 packet and this average diminishes as weight increases for the same reason that average queue diminishes as expected minimum rates increase (see Section IV-B): 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 AQM 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 $S1 - S9$ 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 backoff used by TCP when a timeout occurs and the immediate recovery to a rate of 1 packet per RTT when an acknowledgement arrives. Jumping to 1 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 AVQ+VD.

B. 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 service differentiation. In this subsection we compare the mechanisms used by loss-based approach and this paper. We argue that accumulation-based approach is in principle better than loss-based one in that the former can achieve both steady state objective and 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 $\frac{w-0.5}{w}$ [9]. Nandagopal et al. provide a systematic analysis on how to adjust TCP Reno congestion control increase/decrease parameters to achieve the weighted service [24]. In both work the service is achieved by (adaptively) changing congestion control increase/decrease parameters. This approach creates

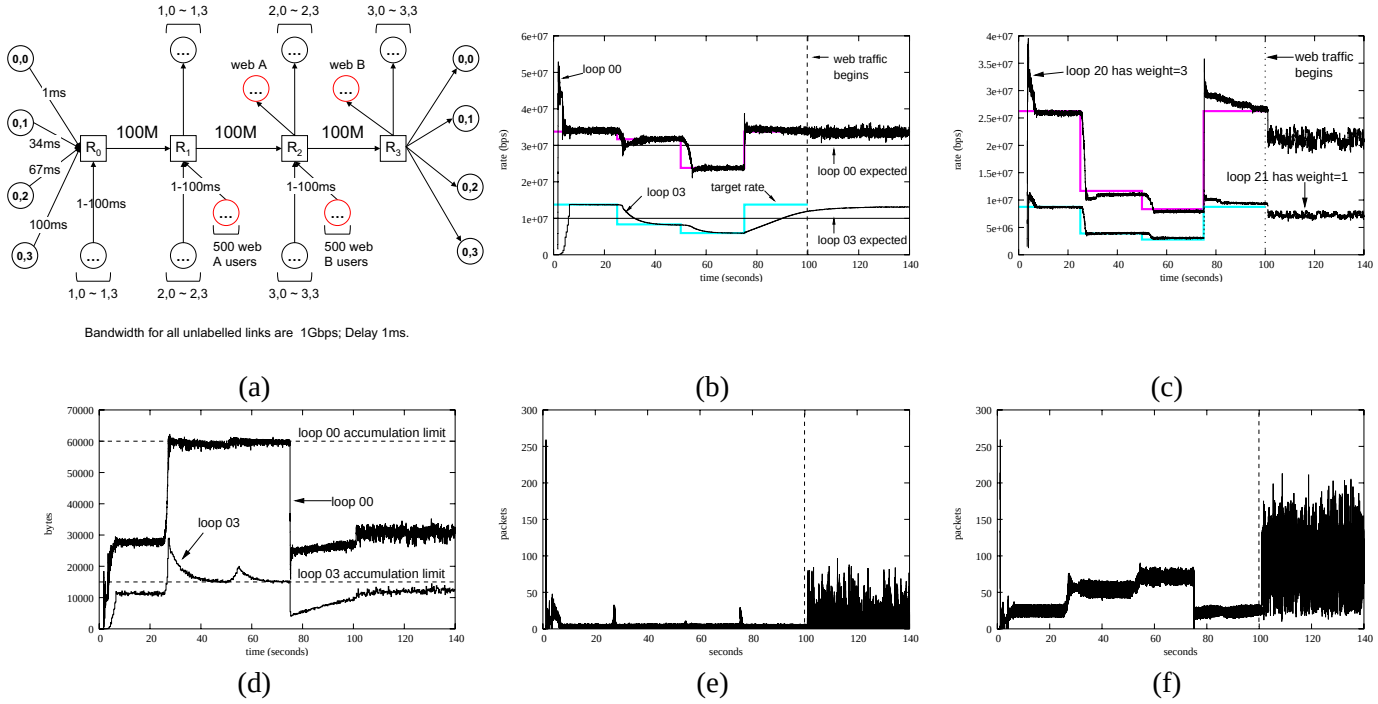


Fig. 6. Simulation of all services co-exist in a complex network of 3 bottlenecks: (a) Topology; (b) Throughput vs. time with rate expectations; (c) Throughput vs. rime with weighted shares; (d) Accumulation vs. time for loops with rate expectations; (e) Bottleneck R_1 queue length (AVQ+VD); (f) Bottleneck R_2 queue length (AQM).

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. 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 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 *relocating* the equilibrium. The control algorithm's dynamic behavior can be designed separately as long as it fits into the general ACC model [30]. 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 at the same time.

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 [24] where it results from the coupling of steady state objective and dynamic performance.

VI. Service Multiplexing

In this section we provide both ns-2 simulation and Linux 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 FIFO and AVQ+VD routers. The simulation and implementation source code will be posted on our project web page soon.

A. Simulations

Firstly we show the simulation results for the topology shown in Figure 6(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. We describe this rate computation momentarily. There are 4 "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 $(1,0) - (1,3)$ are connected to R_0 with propagation delays evenly spread between 1 and 100ms (i.e., 1ms, 34ms, 67ms and 100ms, respectively). Nodes $(2,0) - (2,3)$ have the same delays but are connected to R_1 and likewise for nodes $(3,0) - (3,3)$ to R_2. As specified in Table I, 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 two degrees of oversubscription and back to undersubscribed before a barrage of web-like flows

TABLE I
SIMULATION PARAMETERS

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

start. The *500* web-like flows entering R_1 are evenly distributed across 25 nodes, and each of these nodes is connected to R_1 again with propagation delays evenly spread between 1 and 100ms. The 25 nodes generating web-like bursty traffic to R_2 are similarly connected.

We use simulation due to the rather large number of nodes in our topology. This simulation demonstrates that the target rate allocation is well-defined and controllable via our choice of accumulation limits A_i even under oversubscription.

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 becomes

$$x^*_{ij} = \begin{cases} \frac{w_{ij}}{W}C + x_{ije} & if \;\; i = 0 \\ \\ \frac{w_{ij}}{W_i}(1 - \frac{W_0}{W})C & if \;\; i \neq 0 \end{cases} \quad (18)$$

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.

1) No Oversubscription: As specified in Table I, control loops labelled $(0,0)$ and $(0,3)$ as well as all loops that traverse a single bottleneck start at a random time in $U[0,5]$. The sum of the expected minimum rates for $(0,0)$ and $(0,3)$ is well below the bottleneck capacity, and neither control loop need incur an accumulation greater than its accumulation limit to achieve its expected minimum rate. Thus, as shown in Figure 6(b), both $(0,0)$ and $(0,3)$'s expected minimum rates are satisfied and they both obtain their respective target rates as determined by Equation (18).

2) Accumulation-Limited: At 25s into the simulation, control loop $(0,1)$ begins transmitting and steers toward an expected minimum rate of 35Mbps. The sum of the active expected minimum rates $(30 + 35 + 10)$ is still less than the capacity, but for control loop $(0,3)$ to achieve its expected rate would require $a_{03} > A_{03}$. Therefore, $(0,3)$ becomes bounded by its accumulation limit shown in Figure 6(d) and fails to obtain its expected minimum rate. Notice that even though x_{00e} and x_{01e} are larger than x_{03e}, they are satisfied because we have a policy of giving t them larger accumulation limits.

3) Oversubscription: At 50s, control loop $(0,2)$ 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, control loop $(0,0)$, $(0,1)$, and $(0,2)$ all have similar accumulation limits thereby forcing all to a weighted proportional fair share that satisfies none of the expected minimum rates. This rate allocation is still optimal, which could be validated according to the optimization formulation (10).

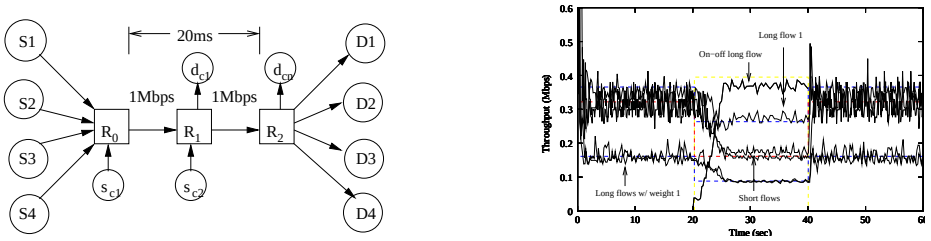
We also note that none of the control loops without an expected rate are starved. Because our system degrades toward weighted proportional fairness, a control loop (i,j) without an expected rate simply receives its weighted share according to w_{ij}.

At 75s, $(0,1)$ and $(0,2)$ stop sending thereby allowing the system to return to an equilibrium that satisfies all expected rates for active control loops.

We note here that throughout the simulation, x_{03} changes slowly compared to x_{00}. This can be attributed to the large difference in round-trip propagation delays; $(0,0)$ has 10ms while $(0,3)$ has 208ms. However, despite their difference in propagation delays these control loops still converge to the appropriate target rate allocation throughout the simulation, or at least until web-like traffic begins at 100 s, at which time the equilibrium is no longer well-defined.

4) Web-like Traffic: At 100s, *500* web users entering bottleneck R_1 and *500* web users entering bottleneck R_2 begin sending and introduce substantial variation in queue lengths, illustrated in Figures 6(e)-(f). The key result is that despite burstiness, each control loop hovers above its expected rate shown in right side of Figure 6(b).

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 than 10%.



(a) (b)

Fig. 7. Implementation of all services co-exist in a network of 2 bottlenecks: (a) Topology; (b) Throughput vs. time.

Control loop $(0, 3)$ does not appear to adjust much in response to bursty web flows. We attribute this to $(0, 3)$'s much longer propagation delay and therefore slower adaptation.

5) Coexisting Bottleneck Mechanisms: AQM and No AQM: As shown in the topology Figure 6(a), bottleneck R_1 uses AVQ+VD while others use FIFO without AQM. Figures 6(e)-(f) readily demonstrate the benefits of AQM. 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: Control loop $(2, 0)$ sends with weight 3 throughout the experiment. For comparison we show its neighboring control loop $(2, 1)$ with weight 1. Because these loops 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 that this happens. Furthermore, Figure 6(c) 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.

B. Implementation Experiments

Besides ns-2 simulations, we also implement Monaco in Linux kernel v2.2.18 [28] 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 2 bottleneck network shown in Figure 7(a) with 1Mbps link bandwidth and 20ms delay. There are 4 long flows which pass all bottlenecks and 2 short 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 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 services. By applying a nonlinear optimization analysis, we show that accumulation could be appropriately manipulated to provide each specific service. We use a set of 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 OS kernel v2.2.18 based on the Click modular router and validate simulation results using the Utah Emulab and an internal testbed.

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

VIII. Acknowledgments

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

References

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

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

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

[4] S. Blake et al. An Architecture for Differentiated Services. *IETF RFC 2475*, Dec 1998.
[5] C. Boutremans, Le Boudec, Jean-Yves. A Note on the Fairness of TCP Vegas. In *Broadband Communications*, 163–170, Feb 2000.
[6] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. *IETF RFC 1633*, Jun 1994.
[7] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465-1480, Oct 1995.
[8] 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.
[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), Jul 1998.
[10] Frame Relay Forum. Http://http://www.frforum.com/.
[11] R. Guérin et al. Scalable QoS Provision Through Buffer Management. In *Proc. SIGCOMM'98*, Sept 1998.
[12] D. Harrison. Edge-to-edge Control: A Congestion Avoidance and Service Differentiation Architecture. *RPI-CS Ph.D. Thesis*, Dec 2001. Available at http://networks.ecse.rpi.edu/˜harrisod/thesis.ps.gz.
[13] V. Jacobson. Congestion Avoidance and Control. In *Proc. SIGCOMM'88*, Aug 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, Aug 2000.
[16] S. Kunniyur and R. Srikant. End-To-End Congestion Control: Utility Functions, Random Losses and ECN Marks. In *Proc. INFOCOM'00*, Mar 2000.
[17] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. In *Proc. SIGCOMM'01*, Aug 2001.
[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, Dec 1999.
[20] S. Low, L. Peterson and L. Wang. Understanding TCP Vegas: A Duality Model. In *Proc. SIGMETRICS'01*, Jun 2001.
[21] L. Massoulie and J. Roberts. Bandwidth Sharing: Objectives and Algorithms. In *Proc. INFOCOM'99*, Mar 1999.
[22] J. Mo and J. Walrand. Fair End-to-End Window-based Congestion Control. *IEEE/ACM Trans. on Networking*, 8(5):556-567, Oct 2000.
[23] J. Mo et al. Analysis and Comparison of TCP Reno and Vegas. In *Proc. INFOCOM'99*, Mar 1999.
[24] T. Nandagopal et al. Scalable Service Differentiation using Purely End-to-End Mechanisms: Features and Limitations. In *IPQoS'00*, Jun 2000.
[25] Network Simulator ns-2. Http://www.isi.edu/nsnam/ns/.
[26] F. Paganini, S. Low, et al. A New TCP Congestion Control with Empty Queues and Scalable Stability. *Submitted*, 2002.
[27] I. Stoica. Stateless Core: A Scalable Approach for Quality of Service in the Internet. *CMU-CS Ph.D. Thesis*, Dec 2000.
[28] A. Venkatesan. An Implementation of Accumulation-based Congestion Control Schemes. *RPI-CS M.S. Thesis*, 2002.
[29] Z. Wang. User-Share Differentiation (USD): Scalable Bandwidth Allocation for Diffrentiated Services. *IETF Draft*, Nov 1997.
[30] Y. Xia, D. Harrison and S. Kalyanaraman et al. Accumulation-based Congestion Control. RPI ECSE Networks Laboratory Technical Report ECSE-NET-2002-12. May 2002. Available at http://www.rpi.edu/˜xiay /pub/acc.ps.gz.

APPENDIX

We provide here the background theory to better understand Sections IV and V for the readers' convenience. Please refer to [30] for a complete description.

Consider the network defined in Section IV-A. Let's firstly analyze from queuing system perspective [22]. 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 II) 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) $\sum_{i \in I_l} x_i \leq c_l$;
(d) $t_{ql} \geq 0$;
(e) $x_i > 0$.

Alternatively, network resource allocation can also be modelled as a nonlinear optimization problem [14] [19] [16]. The network tries to maximize the sum of all flows' utility functions $\sum_{i \in I} U_i(x_i)$, in which flow i's utility function $U_i(x_i)$ is a measure of its happiness when it sends at a rate of $x_i > 0$, subject to a set of capacity constraints $\sum_{i \in I_l} x_i \leq c_l$ at all links. Using Lagrange multiplier method, we construct a Lagrange function $L(\vec{x}, \vec{p}) = \sum_{i \in I} U_i(x_i) + \sum_{l \in L} p_l \cdot \ (c_l - \sum_{i \in I_l} x_i)$. If $U_i(x_i)$ is defined as $U_i(x_i) = s_i \, ln x_i$, where $s_i > 0$ is a weight, then because of the strict concavity of the objective function constrained by a convex set, we apply the Karush-Kuhn-Tucker condition [3] to obtain

Proposition 2: The nonlinear programming problem

$$maximize \qquad \sum_{i \in I} s_i \, ln x_i \tag{19}$$

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

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) $\sum_{i \in I_l} x_i \leq c_l$;
(d) $p_l \geq 0$;
(e) $x_i > 0$.

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

$$U_i(x_i) = a_i\, ln x_i. \tag{20}$$

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 quality of services.

Besides, the Lagrange multiplier p_l is a measure of congestion, or price explored in [19], 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 [17], 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.