Optimization Flow Control, II: Implementation

Sanjeewa Athuraliya Steven H. Low
Department of EEE, University of Melbourne, Australia
{sadsa, slow}@ee.mu.oz.au

Abstract

A duality model of flow control is proposed in Part I of this paper and leads to a basic flow control algorithm. In this sequel we develop a practical implementation of the basic algorithm, Random Exponential Marking (REM). It consists of a link algorithm, that probabilistically marks packets inside the network, and a source algorithm, that adapts source rate to observed marking. REM has three advantages. First the marking probability is exponential in a link congestion measure, so that the *end–to–end* marking probability observed at a source is exponential in its *path* congestion measure. Marking allows the source to estimate its path congestion measure and adjusts its rate in a way that aligns individual optimality with social optimality. Second REM achieves high link utilization with very low backlog, and hence negligible loss and queueing delay. Third sources stabilize around a globally optimal equilibrium, thus avoiding the perpetual cycle of sinking into and recovering from congestion. Moreover the equilibrium can be chosen to achieve different fairness criteria. We present extensive simulation results to demonstrate that REM is not only stable and fair, but more importantly, scalable and robust. Finally, the link algorithm itself can also be used for active queue management that interact with existing source algorithms. We compare the performance of Reno, Reno/RED and Reno/REM.

I. Introduction and summary

A. Motivation

Flow control is a distributed algorithm to share network resources among competing sources. It often consists of two (sub)algorithms: a link algorithm executed inside the network at routers or switches, and a source algorithm executed at edge devices such as host computers or edge routers. The link algorithm detects congestion and feeds back information to sources, and in response, the source algorithm adjusts the rate at which traffic is injected into the network. The basic design issue is what to feed back (link algorithm) and how to react (source algorithm), and the objective is to achieve stability, fairness and robustness. Ideally one should design the link and source algorithm jointly so that they work in concert to steer the network to track a possibly moving desirable operating point. This motivates a recent approach to flow control based on optimization e.g., [6], [14], [16], [12], [18], [23], [19], [1], [13], [25], [26], [17], [3], where the goal is to choose source rates to maximize a global measure of network performance. Flow control, both the link and the source algorithms, is derived as a distributed solution to this welfare maximization problem. Different proposals in the literature differ in their objective function, or solution approach, which lead to different link and source algorithms and their implementation. Though it may not be possible, nor critical, that exact optimality is attained in practice, the optimization framework allows us to understand, and control, the behavior of the network *as a whole*. Indeed we may regard the sources and links as processors in an asynchronous distributed computation system and flow control as a computation to maximize welfare. Under mild conditions on the welfare function, the computation can be proved to converge, i.e., the flow control algorithm is globally stable. Moreover, convergence can be maintained even in an asynchronous environment where sources and links communicate and update at different times, with different frequencies, using outdated information, and feedback delays are different and time-varying [23]. Unlike the works [14], [16], [13], [17] that take a penalty function approach to the solution of the welfare maximization problem, in Part I of this paper we develop a duality approach and derive the flow control as a gradient projection algorithm to solve the dual problem (this will be reviewed in Section II). It is significant that major TCP flow control schemes, Vegas, Reno, Reno/RED, Reno/REM, can all be interpreted within this framework as a dual method [22]; see also [15], [17].

Sanjeewa Athuraliya is supported by the University of Melbourne scholarships.
Steven Low is supported by the Australian Research Council through grants S499705, A49930405 and S4005343.

The basic algorithm of Part I however requires communication between network links and sources that cannot be accommodated on the current Internet. The purpose of this paper is to design and evaluate, through extensive simulations, a practical implementation of the basic algorithm using binary feedback. This is motivated by the recent proposal to introduce Explicit Congestion Notification (ECN) bits in IP (Internet Protocol) headers [10], [28]. A preliminary version of REM is first proposed in [19]; see Section II-C for the difference between this and the current scheme.

B. Random Exponential Marking (REM)

We now summarize the REM algorithm. Detail derivation and justification are given in Section II. A pseudocode implementation is given in Section III.

For our purposes a network is a set L of links with finite capacities $c_l, l \in L$. It is shared by a set S of sources. A source s traverses a subset $L(s) \subseteq L$ of links to the destination, and attains a utility $U_s(x_s)$ when it transmits at rate x_s that satisfies $0 \leq m_s \leq x_s \leq M_s < \infty$. REM is defined by the following link algorithm (1–2) and source algorithm (4–5).

Each link l updates a congestion measure $p_l(t)$ in period t based on the *aggregate* input rate $\hat{x}^l(t)$ and the buffer backlog $b_l(t)$ at link l:

$$p_l(t+1) \;=\; [p_l(t) + \gamma(\alpha_l b_l(t) + \hat{x}^l(t) - c_l)]^+ \tag{1}$$

where $\gamma > 0$ and $\alpha_l > 0$ are small constants and $[z]^+ = \max\{z, 0\}$. Hence $p_l(t)$ is increased when the backlog $b_l(t)$ or the aggregate input rate $\hat{x}^l(t)$ at link l is large compared with its capacity c_l, and is reduced otherwise. Note that the algorithm does not require per–flow information and works with any work conserving service discipline at the link. As we will see in Section II-C, (1) leads to a small backlog ($b_l^* \simeq 0$) and high utilization ($\hat{x}^{*l} \simeq c_l$) at bottleneck links l in equilibrium. Link l marks each packet arriving in period t, that is not already marked at an upstream link, with a probability $m_l(t)$ that is exponentially increasing in the congestion measure $p_l(t)$:

$$m_l(t) \;=\; 1 - \phi^{-p_l(t)} \tag{2}$$

where $\phi > 1$ is a constant. Once a packet is marked, its mark is carried to the destination and then conveyed back to the source via acknowledgement.

The exponential form is critical for a multilink network, because the *end–to–end* probability that a packet of source s is marked after traversing a set $L(s)$ of links is then

$$m^s(t) \;=\; 1 - \prod_{l \in L(s)} (1 - m_l(t)) \;=\; 1 - \phi^{-p^s(t)} \tag{3}$$

where $p^s(t) = \sum_{l \in L(s)} p_l(t)$ is the sum of link congestion measures along the path of source s, a *path* congestion measure. The end–to–end marking probability is high when $p^s(t)$ is large.

Source s estimates this end–to–end marking probability $m^s(t)$ by the *fraction* $\hat{m}^s(t)$ of its packets marked in period t, and estimates the path congestion measure $p^s(t)$ by inverting (3):

$$\hat{p}^s(t) \;=\; -\log_\phi(1 - \hat{m}^s(t)) \tag{4}$$

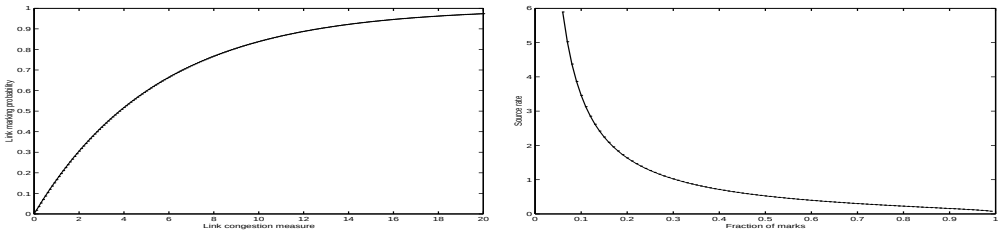
where $\log_\phi$ is logorithm to base ϕ. It then adjusts its rate using marginal utility:

$$x_s(t) \;=\; [U_s^{'-1}(\hat{p}^s(t))]_{m_s}^{M_s} \tag{5}$$

where $U_s^{'-1}$ is the inverse of the marginal utility, $[z]_a^b = \max\{\min\{z, b\}, a\}$. If U_s is strictly concave, then $U_s^{'-1}$ exists and is strictly decreasing. Hence the source algorithm (5) says: if the path $L(s)$ is congested ($\hat{p}^s(t)$ is large), transmit at a small rate, and vice versa.

For example, if $U_s(x_s) = w_s \log x_s$, $x_s \geq 0$, then $x_s(t) = w_s/\hat{p}^s(t)$; if $U_s(x_s) = -(M_s - x_s)^2/2w_s$, $0 \leq x_s \leq M_s$, then $x_s(t) = M_s - w_s\hat{p}^s(t)$ if $\hat{p}^s(t) \leq M_s/w_s$ and 0 otherwise.

The link marking probability (2) and the source rate (5) are illustrated in Figure 1.



(a) Marking probability (b) Source rate

Fig. 1. (a) Marking probability $m_l = 1 - \phi^{-p_l}$ as a function of p_l. (b) Source rate $x_s = U_s^{'-1}(-\log_\phi(1 - \hat{m}^s))$ as a function of $\hat{m}^s$. Here, $\phi = 1.2$ and $U_s(x_s) = 2\log x_s$.

C. Key features of REM

Random Exponential Marking (REM) has three advantages. First it is ideally suited for networks with multiple congested links, where the end–to–end marking probability of a packet incorporates a congestion measure of its *path*. This allows a source to estimate its path congestion measure by observing the fraction of its packets that are marked. The use of marking as a means for sources to estimate information on their paths seems novel and applicable in other contexts. Second, by equalizing input rate $\hat{x}^{*l}$ with capacity c_l and driving backlog b_l^* to zero, the update rule (1) leads to very high utilization with negligible loss or queueing delay. Third, as we will see, under REM, the sources and links can be thought of as carrying out a stochastic approximation algorithm to maximize the aggregate utility $\sum_s U_s(x_s)$ over the source rates x_s, $s \in S$, subject to link capacity constraints. As alluded to earlier, it is not critical that optimality is exactly attained in practice. It is however significant that REM attempts to steer the network as a whole towards a desirable operating point. Moreover this operating point can be chosen to achieve desired fairness.

We have done extensive simulations to evaluate four properties of REM: stability, fairness, scalability and robustness. We now summarize our findings.

REM can be regarded as a stochastic version of the basic algorithm in [18], [23]. Though we have not proved analytically the stability and fairness of REM, our simulation results confirm that it inherits the stability and fairness properties of the basic algorithm. It is proved that the basic algorithm converge to the unique optimal that maximizes aggregate source utility even in an asynchronous environment [23, Theorems 1 and 2]. Moreover, the equilibrium can be chosen to achieve different fairness criteria, such as proportional [14] or maxmin fairness, by appropriate choice of source utility functions [23, Theorems 3 and 4]. Simulation results in later sections show that REM converges quickly to a neighborhood of the equilibrium, and then fluctuates around it. Hence the basic algorithm determines the macroscopic behavior of REM, including stability and fairness.

A focus of our simulation study is to explore the scalability and robustness of REM. There are two aspects of scalability: complexity and performance. Both the link algorithm (1–2) and the source algorithm (4–5) use *only* local, and aggregate, information. Their complexity does not increase with the number of sources or the number of links or their capacities. Moreover they do not need to be restarted as network conditions, such as the link capacities, the set of sources, their routes or utility functions, change. Hence REM is applicable in a dynamic network even though it is derived from a static model. A critical issue however is whether performance scales. We present simulation results to demonstrate that REM's performance, such as throughput, utilization, queue length

and loss, remains stable when traffic load, link capacity, propagation delay, or network size is scaled up by a factor of 10.

We evaluate robustness both with regard to parameter setting and to modeling assumptions. First REM is characterized by three main parameters: γ that determines the rate of convergence, α_l that trades off link utilization and delay, and ϕ that affects the marking probability. The scalability experiments also demonstrate REM's robustness to parameter setting, i.e., its performance remains stable in an environment that is drastically different from the nominal environment with respect to which the parameter values have been chosen. Second REM estimates round trip time in order to translate rate to window control. Simulations indicate that REM is robust to error in round trip time estimation.

D. Structure of paper

In Section II we first review the optimization framework and the basic flow control algorithm developed in [18], [23]. We then derive REM by simplifying the communication requirement of the basic algorithm. In Section III we describe our simulation setup and pseudocode. We then present simulation results on stability and fairness of REM in Section IV and on scalability and robustness in Section V. In Section VI we discuss how to set parameters in REM. We also contrast REM with TCP Reno, Vegas and RED, and argue that the fundamental difference between them is the way congestion is measured and that this difference is the underlying reason for the key features of REM. The link algorithm of REM can interwork with current TCP schemes, and in Section VII, we present simulation results to compare the performance of Reno (with DropTail), Reno/RED and Reno/REM. We conclude in Section VIII with some applications and limitations of this work.

II. Derivation of REM

A. Model

Consider a network that consists of a set L of unidirectional links of capacity c_l, $l \in L$. The network is shared by a set S of sources. Source s is characterized by four parameters $(L(s), U_s, m_s, M_s)$. The path $L(s) \subseteq L$ is a set of links that source s uses, $U_s : \Re_+ \to \Re$ is a utility function, $m_s \geq 0$ and $M_s < \infty$ are the minimum and maximum transmission rates, respectively, required by source s. Source s attains a utility $U_s(x_s)$ when it transmits at rate x_s that satisfies $m_s \leq x_s \leq M_s$. We assume U_s is strictly concave increasing and twice continuously differentiable in its argument. For each link l let $S(l) = \{s \in S \mid l \in L(s)\}$ be the set of sources that use link l. By definition $l \in L(s)$ if and only if $s \in S(l)$.

Our objective is to choose source rates $x = (x_s, s \in S)$ so as to:

$$\max_{m_s \leq x_s \leq M_s} \quad \sum_s U_s(x_s) \tag{6}$$

$$\text{subject to} \quad \sum_{s \in S(l)} x_s \ \leq \ c_l, \quad l = 1, \ldots, L \tag{7}$$

The constraints (7) say that the aggregate source rate at any link l does not exceed the capacity. A unique maximizer, called the *(primal) optimal* rates, exists since the objective function is concave, and hence continuous, and the feasible solution set is compact.

Solving the primal problem (6–7) directly is impractical over a large network since it may require coordination among possibly all sources due to coupling through shared links. The key to a distributed and decentralized solution can be obtained by looking at its dual [23].

Associated with each link l is a dual variable p_l. The dual problem of (6–7) is to choose the dual vector $p = (p_l, l \in L)$ so as to

$$\min_{p \geq 0} \quad D(p) \ := \ \sum_s B_s(p^s) + \sum_l p_l c_l \tag{8}$$

where

$$B_s(p^s) = \max_{x_s \geq 0} U_s(x_s) - x_s p^s \quad (9)$$

$$p^s = \sum_{l \in L(s)} p_l \quad (10)$$

If we interpret the dual variable p_l as the price per unit bandwidth at link l, then p^s in (10) is the price per unit bandwidth in the path of s. Hence $x_s p^s$ in (9) is the bandwidth cost to source s when it transmits at rate x_s, $U_s(x_s) - x_s p^s$ is the net benefit of transmitting at rate x_s, and $B_s(p^s)$ is the maximum benefit s can achieve at the given (scalar) price p^s. A vector $p \geq 0$ that minimizes the dual problem (8) is called *dual optimal*. Given a vector price $p = (p_l, l \in L)$ or a scalar price $p^s = \sum_{l \in L(s)} p_l$, we will abuse notation and denote the unique maximizer in (9) by $x_s(p)$ or by $x_s(p^s)$.

There are two important points to note. First, given scalar prices p^s, sources s can easily solve (9) to obtain the individually optimal source rates $x(p) = (x_s(p^s), s \in S)$ without having to coordinate with any other sources. Indeed by the Karush–Kuhn–Tucker theorem, we have

$$x_s(p^s) = [U_s^{'-1}(p^s)]_{m_s}^{M_s} \quad (11)$$

where $[z]_a^b = \min\{\max\{z, a\}, b\}$. Here $U_s^{'-1}$ is the inverse of U_s', which exists over the range $[U_s'(M_s), U_s'(m_s)]$ when U_s is continuously differentiable and *strictly* concave. Second, by duality theory, there exists a dual optimal price $p^* \geq 0$ such that these individually optimal rates $x^* = (x_s(p^{*s}), s \in S)$, i.e., each $x_s(p^{*s})$ solves (9), are also socially optimal, i.e., solve (6–7) as well. Furthermore, as we will see below, solution of the dual problem can be distributed to individual links and sources. Hence a better alternative to solving the primal problem (6–7) directly is to solve its dual (8) instead.

In the rest of the paper, given a price (vector) p, we will refer to p_l as *link price* and $p^s = \sum_{l \in L(s)} p_l$ as *path price* of source s. It can be interpreted in two ways. First, the price p is a congestion measure at the links: the larger the link price p_l, the more severe the congestion at link l. The path price p^s is thus a congestion measure on the path of source s. Second, an *optimal* p^* is a shadow price (Lagrange multiplier) associated with the constrained maximization (6–7); i.e., p_l^* is the marginal increment in aggregate utility $\sum_s U_s(x_s)$ for a marginal increment in link l's capacity c_l. We emphasize however that p may be unrelated to the actual charge users pay. If sources are indeed charged according to these prices, then p^* aligns individual optimality with social optimality, thus providing the right incentive for sources to choose the optimal rates.

B. Basic algorithm

The dual problem is solved in [18], [23] using gradient projection method (e.g., [24], [4]) where link prices are adjusted in opposite direction to the gradient $\nabla D(p(t)) = (\partial D / \partial p_l(p(t)), l \in L)$:

$$p_l(t+1) = [p_l(t) - \gamma \frac{\partial D}{\partial p_l}(p(t))]^+$$

Here $\gamma > 0$ is a stepsize, and the gradient is given by

$$\frac{\partial D}{\partial p_l}(p(t)) = c_l - x^l(p(t))$$

Hence the price computation can be distributed to each individual link. Indeed the algorithm takes the familiar form of flow control: in each iteration t, each link l individually updates its own price $p_l(t)$ based on the *aggregate* rate at link l, and each source s individually adjusts its rate based on its path price $p^s(t)$.

To describe it precisely, we abuse notation and use $x_s(\cdot)$ both as a function of time t and a function of price $p(t)$ given by (11); the meaning should be clear from the context. Let $x^l(t) = \sum_{s \in S(l)} x_s(t)$ represent the aggregate

source rate at link l at time t, and $p^s(t) = \sum_{l \in L(s)} p_l(t)$ represent the path price of source s at time t. Then the price computation (link algorithm) and rate adjustment (source algorithm) are given by:

Basic algorithm:

$$p_l(t+1) = [p_l(t) + \gamma(x^l(t) - c_l)]^+, \ l \in L \tag{12}$$

$$x_s(t+1) = x_s(p^s(t)), \quad s \in S \tag{13}$$

In (12), $x^l(t)$ represents the demand for bandwidth at link l and c_l represents the supply. The price is adjusted according to the law of demand and supply: if demand exceeds supply, raise the price; otherwise reduce it. In (13), $x_s(p^s(t))$ is referred to as the demand function in microeconomics: the higher the path price $p^s(t)$ (i.e., the more congested the path), the lower the source rate.

It is proved in [23] that the basic algorithm (12–13) converges to the unique optimal rates provided the utility functions are strictly concave increasing, their second derivatives are bounded away from zero, and the stepsize $\gamma > 0$ is sufficiently small. Specifically if $\{(x(t), p(t))\}$ is a sequence generated by (12–13) then any limit point (x^*, p^*) is primal–dual optimal. Moreover, provided that the sources and links perform their updates frequently enough, convergence is maintained even in an asynchronous environment where sources and links may compute and communicate at different times with different frequencies, and where feedback delays are substantial and time–varying.

It is also proved there that different utility functions can be chosen to achieve different fairness criteria on the optimal rates.

C. Implementation: REM

Under the basic algorithm (12–13) a link l needs the aggregate source rate $x^l(t)$ for price computation and a source s needs feedback of a scalar price $p^s(t)$ for rate adjustment. This communication requirement cannot be accommodated on the current Internet. In this subsection, we first explain how to perform price computation based on input rate and buffer occupancy locally at a link, thus eliminating the need for explicit communication from sources to links. We then describe how to feed back the prices to sources using only a single bit. The combination is the REM algorithm introduced in Section I-B. As we see below, the price computation rule (PC3) here is different from those in [21], [19] (PC1 and PC2), and the difference is critical in achieving very high utilization with ngeglible backlog or loss.

C.1 Price computation

Notice that $x_s(t)$ is the *source* rate and is generally different from the input rate at a link $l \in L(s)$ from source s, unless the link is the first in the path of source s, because the fluid flow is modified as it passes through successive links. Let $x_{ls}(t)$ be the input rate from source s at link l at time t, and $\hat{x}^l(t) = \sum_{s \in S(l)} x_{ls}(t)$ be the *aggregate input* rate at link l. The aggregate input rate $\hat{x}^l(t)$ is generally different from the aggregate source rate $x^l(t) = \sum_{s \in S(l)} x_s(t)$ used in the basic algorithm. They are equal *in equilibrium* when buffer stabilizes [21]. We assume each link has a large buffer so that no packets are lost. Let $b_l(t)$ be the (aggregate) buffer backlog at link l at time t. Then $b_l(t)$ evolves according to:

$$b_l(t+1) = [b_l(t) + \hat{x}^l(t) - c_l]^+. \tag{14}$$

Both the aggregate input rate $\hat{x}^l(t)$ and the backlog $b_l(t)$ can be measured at link l.

We now present three algorithms for price computation. All three are based on the idea of approximating the gradient $\nabla_l D(t) = c_l - x^l(t)$ in carrying out the gradient projection algorithm (12) using local information. This eliminates the need for sources to communicate their rates to links in their paths.

The first algorithm approximates the gradient $c_l - x^l(t)$ by estimating the aggregate source rate $x^l(t)$ by the aggregate input rate $\hat{x}^l(t)$ (cf. (12)):

$$\textbf{PC1:} \qquad p_l(t+1) \;=\; [p_l(t) + \gamma(\hat{x}^l(t) - c_l)]^+$$

Multiplying both sides of (14) by the positive stepsize γ, we see that the buffer process automatically performs the price computation PC1, *provided that c_l is the true link capacity available to serve the sources in $S(l)$* and that we identify $p_l(t)$ with $\gamma b_l(t)$. Our second algorithm thus simply sets the price to a fraction of the buffer occupancy:

$$\textbf{PC2:} \qquad p_l(t) \;=\; \gamma b_l(t)$$

PC2, originally proposed in [21], is simpler to implement as links do not need to measure the aggregate input rate. It however does not scale: as the number of sources increases, the equilibrium price vector p^*, and hence the equilibrium buffer vector $b^* = \gamma^{-1}p^*$, increases steadily. This not only necessitates large buffer in the network, but worse still, it leads to large feedback delays. Algorithm PC1, used in [19], can alleviate the problem by setting c_l in PC1 to be a fraction $\rho \in (0,1)$ of the true link capacity. Then in equilibrium, the input rate $\hat{x}^l(t) = c_l$ is strictly less than the true link capacity and hence backlogs will clear. However, to be effective, ρ needs to be significantly less than 1, leading to low utilization. These will be illustrated in the simulation results below.

These considerations motivate our third algorithm (the one introduced in Section I):

$$\textbf{PC3:} \qquad p_l(t+1) \;=\; [p_l(t) + \gamma(\alpha_l b_l(t) + \hat{x}^l(t) - c_l)]^+$$

where c_l can be the true link capacity. Here $\alpha_l > 0$ is a small constant that can be different at different links. In equilibrium, price p^* stabilizes. For a nonbottleneck link with $p_l^* = 0$, backlog is zero $b_l^* = 0$ and $\hat{x}^{*l} \le c_l$. For a bottleneck link with $p_l^* > 0$, we must have $\alpha_l b_l^* + \hat{x}^{*l} = c_l$. If the equilibrium buffer is nonzero $b_l^* > 0$, then the input rate is strictly less than the capacity $\hat{x}^{*l} < c_l$, and hence the buffer b_l^* could not have been in equilibrium. Hence, by contradiction, we must have both zero buffer $b_l^* = 0$ and full utilization $\hat{x}^{*l} = c_l$ in equilibrium, provided prices are fed back exactly to sources. When prices are fed back only approximately using a single bit, as in REM, the source rates and backlogs fluctuate around their equilibrium values. The random fluctuation can be attributed to noise and delay associated with estimation of path prices by the sources from marked packets. See the simulation results in Section IV-A.

The term $\hat{x}^l(t) - c_l$ in PC3 equalizes input rate with capacity and the term $b_l(t)$ empties the buffer. We can replace $b_l(t)$ by a general function $f_l(b_l(t))$. For example, in the simulations in Section VII on Reno/REM, we have used $f_l(b_l(t)) = b_l(t) - b_{l0}$. This has the effect of stabilizing the equilibrium buffer around $b_{l0} > 0$ in order to attain higher utilization during transient.

We prove in [21] that, when (12) is replaced by PC1 or PC2, the price updates are still in the descent direction, i.e., $D(t+1) < D(t)$, provided the stepsize γ is sufficiently small. This implies that the error in gradient estimation converges to zero and the algorithm converges to yield the optimal rates. We are however unable to prove analytically the convergence of PC3, except in the single–link case. Difficulty arises because PC3 is no longer a descent algorithm. It has however always converged in all our simulation experiments.

C.2 Price feedback

To feedback prices using a single bit, the basic idea is for a source s to estimate the path price $p^s(t)$ from packet marking and adjust its rate according to (13) using the estimate $\hat{p}^s(t)$ in place of the true value $p^s(t)$. This is first proposed in [19]. We now describe the method for price feedback in an abstract synchronous model where time is slotted into update periods. Sources and links update their prices and rates at the beginning of each period.[1]

[1] In the simulation below, however, sources estimate their end–to–end marking probabilities *asynchronously* on the arrival of every acknowledgement based on marks in the past N acknowledgements; see the pseudocode in Section III.

On packet arrival in period t, if it is not marked, a link l marks it with probability $m_l(t)$ given by (2), independent of all other packets. Hence the higher the price the more likely packets are marked. The end–to–end marking probability for packets of source s is then $m^s(t)$ given by (3). A mark is placed in the ECN bit of a packet enroute to its destination and is carried back to its source in the ECN bit of the packet's acknowledgement, unmodified in the return path.

A source estimates $m^s(t)$ by the fraction of marked packets in period t. Suppose source s receives acknowledgement for packets $1, 2, \ldots, N(t)$ in period t. Let $E_k(t)$ be 1 if the kth packet in period t is marked and 0 otherwise, $k = 1, 2, \ldots, N(t)$. Let $\hat{m}^s(t)$ be an estimate of the end–to–end marking probability $m^s(t)$:

$$\hat{m}^s(t) \quad = \quad \frac{1}{N(t)} \sum_{k=1}^{N(t)} E_k(t)$$

Then we obtain a price estimate $\hat{p}^s(t)$ through (4). The estimate is used to determine a new source rate through (5), in place of the true path price $p^s(t)$ in the basic algorithm.

Putting the price computation and price feedback method together yields the REM algorithm in Section I-B.

D. Smoothed REM

We have found from simulations that a smoothed version of REM performs better especially when the end–to–end marking probabilities in the network take extreme values (close to 0 or 1); see Section VI. In smoothed REM, a source adjusts its window once every round trip time. For each adjustment, the window is incremented or decremented by 1 (or a small fraction, say, 10%, of the current window size) according as the target value determined by the price is larger or smaller than the current window. This is in the spirit of Vegas [5].

III. Simulation setup

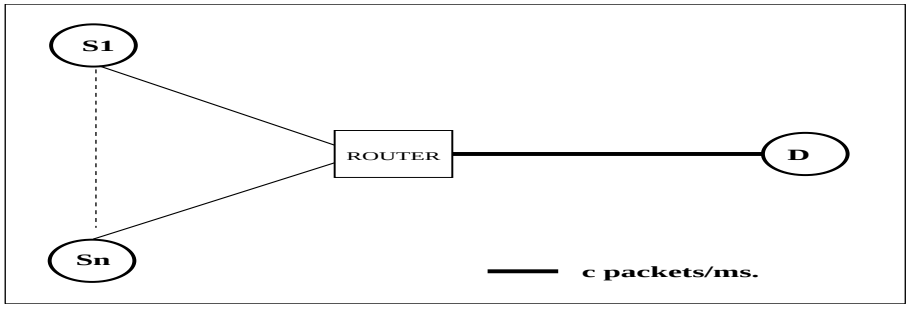
We list here the network topology, source parameters, and link parameters that are common to most simulations. Other details that may vary across simulations will be given in the following subsections.

All simulations are conducted for one of the two networks shown in Figure 2. The single (bottleneck) link network consists of n sources transmitting to a common destination. Each source is connected to a router via an access link and then to the destination via a shared link. In the multilink network, only the shared links are shown, not the access links. There are n shared links all with the same capacity, one long connection using all the n links and n short connections each using a single link as shown in the figure. This network is widely used in previous studies, e.g., in [9]. In both networks the shared link(s) has (have) a lower capacity than the access links and is (are) the only bottleneck(s). At each link packets are served in FIFO order.

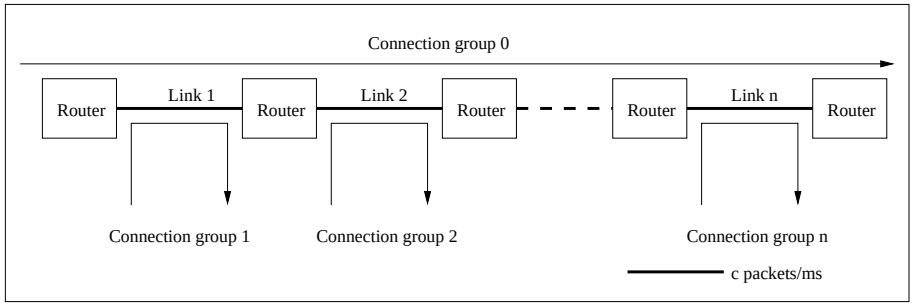
The utility functions of the REM sources are $w_s \log x_s$, where w_s may take different values in different experiments. The source rate is controlled through windowing, where the rate calculated at a source is converted to a window size by multiplying it by estimated round trip time. The sources are greedy and always exhaust the window. Destination immediately sends an acknowledgement on receipt of a packet. The maximum rate M_s for each source is $2 \times c$ where c in packets/ms is the bottleneck link capacity. The minimum source rate m_s is 0.1 packets/ms.

Our discrete time packet–level simulations are written in MATLAB. The pseudocode are given in Figure 3 for link algorithm and Figure 4 for source algorithm. We make two remarks. First we initialize fraction of marks in the source algorithm to 1 so that window starts at its minimum and increases gradually[2]. Second the variable *fraction* in Figure 4 is updated on each ACK arrival as follows. Let *earliest* denote the variable which is 1 if the earliest of the last N ACKs is marked, and 0 otherwise. Let *mark_of_ACK* denote the mark on the newly arrived ACK. Then, *fraction* is updated thus:

[2]Since we focus on the equilibrium situation in this paper, sources in our simulations increases their rates *linearly* from their initial minimum. An alternative is to increase *exponentially* as in the slow–start phase of TCP.



(a) Single (bottleneck) link network



(b) Multilink network

Fig. 2. Network topologies. In the single-link network, propagation delays vary across simulations. In the multilink network, short connections each has round trip propagation delay of 7ms, long connection $2n + 3$ms.

---

for each ACK arrival
    if $(earliest = 1)$ and $(mark\_of\_ACK = 0)$
        $fraction \leftarrow fraction - \frac{1}{N}$
    elseif $(earliest = 0)$ and $(mark\_of\_ACK = 1)$
        $fraction \leftarrow fraction + \frac{1}{N}$
    endif
endfor

Unless otherwise specified, the parameter values are (refer to pseudocode): $\delta = 0.1$, $\beta = 0.01$, $N = 100$, $\alpha_l = 0.1$, $\gamma = 0.005$. The value of ϕ varies; see the following sections.

IV. Stability and fairness

In this section we present simulation results to confirm that it inherits the stability and fairness properties of the basic algorithm: REM tracks the behavior of the basic algorithm with added oscillations. Through appropriate choice of utility functions, different fairness criteria can be enforced, in equilibrium, regardless of the propagation delays of the sources. The results also confirm that PC3 is superior to PC1 and PC2. Hence the rest of the simulations all use PC3.

A. Stability

Simulation has been conducted on the single–link network in Figure 2(a) with four sources. The capacity c of the bottleneck link is 12 packets/ms. The parameter $\phi = 1.4$. The round trip propagation delays of the sources are

periodically
  update aggregate input rate:
    $in \leftarrow (1-\delta) \times in + \delta \times new\_in$
  udpate marking probability m_l**:**
    $p_l \leftarrow \max\{p_l + \gamma(\alpha_l \times buffer + in - capacity),\ 0\}$
    $m_l \leftarrow 1 - \phi^{-p_l}$
endperiodically

while buffer not empty
  mark packet with probability m_l as it leaves
endwhile

Saved variables:
  in: aggregate input rate estimate
  p_l: link price
  m_l: current marking probability
Fixed parameters:
  δ: weight in aggregate input rate estimation
  γ: stepsize in price adjustment
  α_l: weight of buffer in price adjustment
  ϕ: base in marking probability computation
Temporary variables:
  new_in: current aggregate input rate
  $buffer$: current buffer occupancy (may be smoothed)
  $capacity$: current link capacity (may be estimated)

Fig. 3. Pseudocode for link algorithm

7, 9, 11, 13 ms. The utility function of the REM sources is $w_s \log x_s$, where, for all s, $w_s = 12$ packets/ms. The starting times of the sessions are staggered by 2 s. S1 has been active prior to time $t = 0s$ and remains active for the entire simulation. The second source, S2, is active from 0s-10s, S3 from 2s-8s and S4 from 4s-6s.

Simulation results for PC1, PC2, and PC3 are shown in Figures 5–7. We first describe some common features and then contrast their differences.

The straight lines in the figures show the theoretical equilibrium values if basic algorithm were used. The window sizes exhibit more severe oscillation initially when only few (two) sources are active. This is because the equilibrium price p^* of the bottleneck link is small then and, for log utility function, the source rate $x^* = x_s(p^*) = w_s/p^*$ is sensitive to changes in p^* when p^* is small (see Figure 1(b)). As new sources become active, p^* increases and x^* becomes less sensitive.

The fluctuation around the equilibrium values can be attributed to the use of binary feedback and to the translation from rate to window control. The binary feedback introduces both noise and delay into the price estimation at the sources. The translation from rate to window control relies on estimation of the current round trip time, which introduces noise and delay into the price computation at the links. Despite these factors, REM tracks the behavior of the basic algorithm, and in this sense, it seems stable.

We now compare the performance of PC1, PC2 and PC3.

A.1 PC1

Figure 5 shows the simulation results for PC1. Unlike all other simulations presented, c_l is set to 85% of the true link capacities to avoid buffer buildup. The buffer level exhibits brief spikes when a source activates. Because the target utilization is 85%, the buffer tends to a neighborhood of zero in the steady state. Notice that the largest and smallest round trip bandwidth–delay product to the bottleneck link are 156 and 84 packets, respectively, while

for each ACK arrival
  update round trip time estimate:
    $RTT \leftarrow (1-\beta) \times RTT + \beta \times RTT\_of\_ACK$
  update *fraction* **of marks in the last** N **ACKs**
  calculate new rate:
    if $fraction = 0$
      $x_s \leftarrow max\_rate$
    elseif $fraction = 1$
      $x_s \leftarrow min\_rate$
    else
      $p^s \leftarrow -\frac{\log(1-fraction)}{\log \phi}$
      $x_s \leftarrow \max\{\min\{w_s/p^s,\ max\_rate\},\ min\_rate\}$
    endif
  set window size:
    $window \leftarrow$ ceiling $(x_s \times RTT)$
endfor

Saved variables:
  RTT: round trip time estimate
  fraction: fraction of marks in last N ACKs
  window: window size
Fixed parameters:
  β: weight in RTT estimation
  N: sample size for price estimation
  ϕ: base in price estimation
  max_rate: maximum source rate
  min_rate: minimum source rate
Temporary variables:
  RTT_of_ACK: round trip time of new ACK
  p^s: path price
  x_s: source rate

Fig. 4. Pseudocode for source algorithm with $U_s(x_s) = w_s \log x_s$.

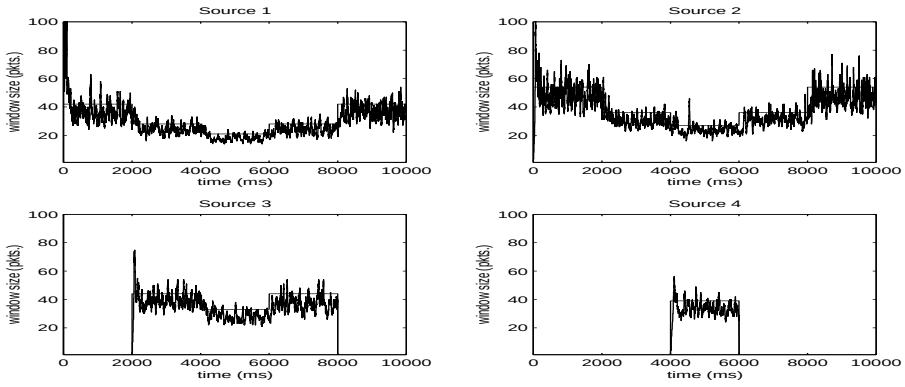
the peak backlog is only about 90 packets. The disadvantage of this scheme is low utilization.

A.2 PC2

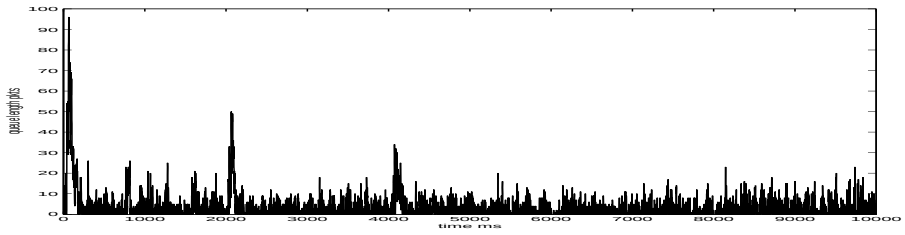
Figure 6 shows that congestion windows and buffer occupancy oscillate more severely under PC2. This is because the stepsize $\gamma = 0.01$ is two times larger than that for PC1. A large γ has been chosen to control the size of the equilibrium buffer occupancy. This highlights a major disadvantage of this scheme: the equilibrium backlog $b_l^* = p_l^*/\gamma$ increases with the equilibrium price p_l^*. Since equilibrium price p_l^* increases with the number of active sources, we need a large γ to maintain a low equilibrium backlog b_l^*. Notice the steady increase in average backlog in Figure 6(b) as sources activate. A large γ however leads to severe oscillation and even divergence of prices and rates ([23, Theorem 1]). A small γ, on the other hand, yields a large equilibrium backlog and increases the round trip time, leading to lag–induced oscillation. The utilization is however the highest ($> 98\%$) among the three algorithms due to a much larger backlog (notice the different scales).

A.3 PC3

Figure 7 shows the simulation results for PC3. As in PC1, the congestion windows and the prices rapidly converge to a neighborhood of their equilibrium values after each disturbance, while the backlog oscillates about zero. Again, the average backlog is a small fraction of the round trip bandwidth–delay product to the bottleneck link. Unlike PC1, the bottleneck link is better utilitized (c_l is set to true link capacities).



(a) Window size



(b) Buffer occupancy

Fig. 5. Stability – PC1.

All simulations presented below use PC3 because of its superiority over PC1 and PC2.

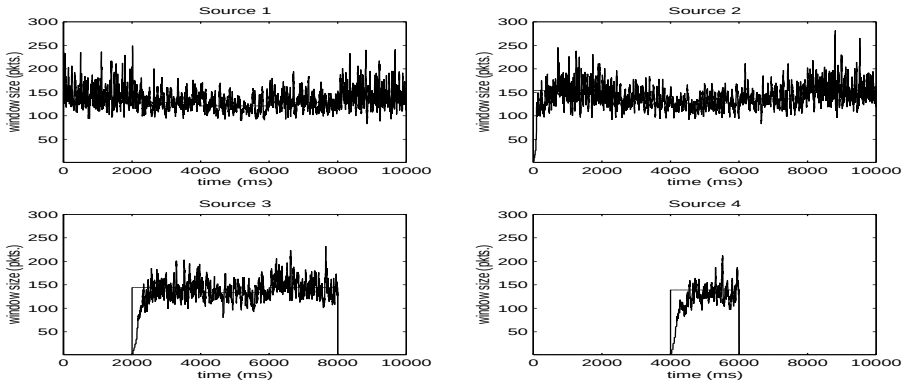
Table I summarizes the link utilization and equilibrium backlog under PC1, PC2, and PC3 for each 2–second period.

B. **Fairness**

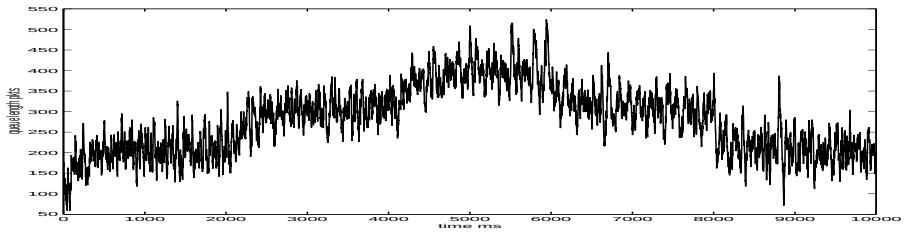
The parameters w_s in utility functions $U_s(x_s) = w_s \log x_s$ determine the relative share of bandwidth each source receives in equilibrium. In this section we present three sets of results to demonstrate that by appropriate choice of w_s it is possible to achieve different fairness criteria regardless of the propagation delay of the sources; cf. [23, Theorem 4].

The first set of results is for the single–link network in Figure 2(a) with the same topology and parameter values as in the stability experiments. The parameters w_s have been set to $c/\left(d_s \sum_{s \in S} 1/d_s\right)$ where $c = 12$ packets/ms is the bottleneck link capacity and d_s is the round trip propagation delay of source s, so that all sources would have the same equilibrium window size (but different rate) under the basic algorithm. Source 1 has been active before time 0s. Source 2 turns on at time 0s, source 3 at time 1s, and source 4 at time 2s. Once active each source continues to transmit for the duration of the simulation. Figure 8(a) gives the simulation results. As expected, the windows settle around the common equilibrium value.

The second set of simulations are for the multilink network in Figure 2(b) where all sources have equal $w_s = 12$ packets/ms. This implies that the equilibrium rates should be proportionally fair. Figure 8(b) shows the results of 10 experiments with networks of sizes $1, \ldots, 10$ links. For a network of n links, we measure the throughput share of the long connection at each of the n links, $x_0^*/(x_0^* + x_i^*)$, $i = 1, \ldots, n$, where x_i^* are the equilibrium rates of connections i, $i = 0, 1, \ldots, n$. It is represented by each bar in the figure. As the network size increases the long



(a) Window size



(b) Buffer occupancy

Fig. 6. Stability – PC2.

connection sees a higher price and hence its share steadily decreases. The measured share matches very well the expected value.

Similar results for maxmin fairness is presented in the next section for scalability experiments.

V. Scalability and robustness

In this section we present experimental results on scalability and robustness. As discussed in Introduction the REM algorithm, (1–2) and (4–5), involves only local and aggregate information, and hence its complexity scales. A critical issue however is whether its *performance* remains stable as we scale up the number of sources, the link capacity, the propagation delay, or the network size. We present four experiments to demonstrate that it does.

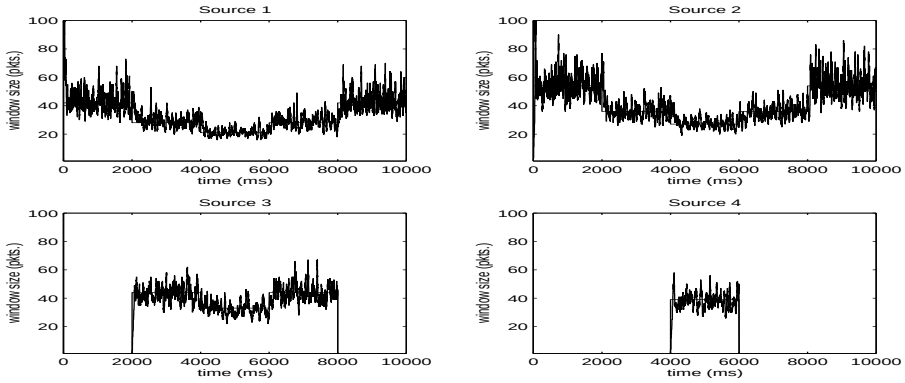
The scalability experiments also show that REM performs well across a wide range of network conditions. This makes tuning of its parameters easier. We present, in addition, two experiments to demonstrate REM's robustness to packet loss and to errors in round trip time estimation.

A. Scalability

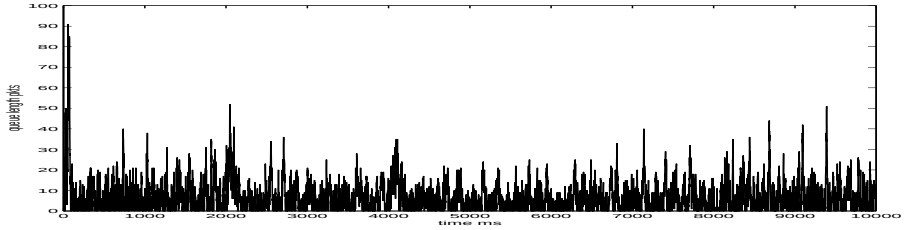
A.1 Traffic load

This set of 10 experiments shows that REM copes well as traffic load increases. Each experiment uses the single–link network of Figure 2(a) with n sources, $n = 10, 20, ..., 100$. All sources have the same utility function $U_s(x_s) = 12.5 \log x_s$ and round trip propagation delay of 10ms. The bottleneck link has a capacity of 25 packets/ms and a finite buffer of size 50 packets. The equilibrium price for the nth experiment, with n sources, is thus $p^*(n) = n/2$. For all the 10 experiments, the REM parameters are: $\gamma = 0.001$, $\alpha_l = 0.1$, $\phi = 1.05$.

The results are shown in Figure 9. The equilibrium source rate, averaged over all sources, decreases steadily as

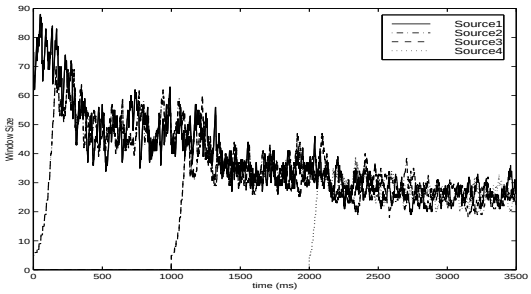


(a) Window size

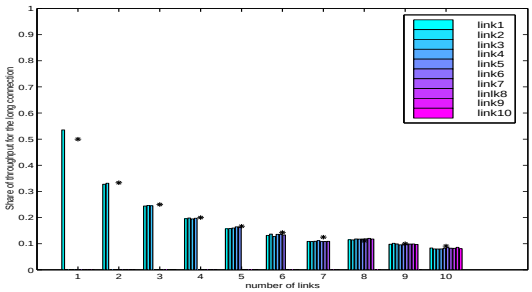


(b) Buffer occupancy

Fig. 7. Stability – PC3.



(a) Same window size



(b) Proportional fairness

Fig. 8. Fairness. For proportional fairness, each bar represents the throughput share of the long connection at one of the links and star '*' represents the theoretical value.

	Time interval	0–2s	2–4s	4–6s	6–8s	8–10s
PC1	Utilization	86%	86%	86%	84%	84%
	Avg. backlog (pkts)	4.8	2.6	2.4	1.5	2.3
PC2	Utilization	100%	100%	100%	100%	100%
	Avg. backlog (pkts)	196.9	293.5	386.4	313.3	209.9
PC3	Utilization	94%	96%	96%	95%	94%
	Avg. backlog (pkts)	7.8	6.5	5.8	5.7	6.6

TABLE I

COMPARISON OF PC1, PC2, PC3.

the number of sources increases and matches well the theoretical value. The equilibrium link utilization remains above 96% while the equilibrium loss ($< 0.2\%$) and backlog (< 10 packets) remains low.

A.2 **Capacity**

This set of 10 experiments are similar to the previous set, except that the number of sources is fixed at 20 but the link capacity is increased from 10 to 100 pkts/ms at 10 pkts/ms increment. The round trip propagation delay is 10ms and the buffer size is 40 pkts. REM parameters for all 10 experiments are: $\gamma = 0.005$, $\alpha_l = 0.1$, $\phi = 1.1$.

The results are shown in Figure 10. The equilibrium source rate, averaged over all sources, increases linearly as link capacity increases and matches well the theoretical value. The equilibrium link utilization remains above 96% while the equilibrium loss ($< 1\%$) and backlog (< 14 packets) remains low.

A.3 **Propagation delay**

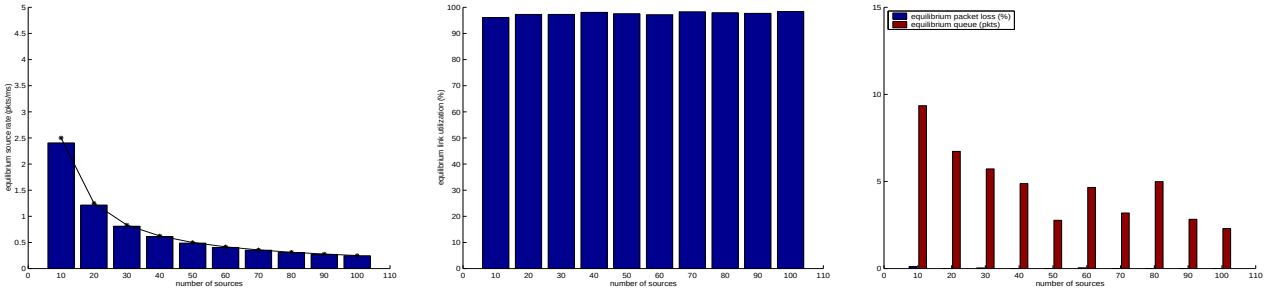
This set of 10 experiments are similar to the previous set, except that the link capacity is fixed at 20 pkts/ms but the round trip propagation delay is increased from 10 to 100 ms at 10 ms increment. For all 10 experiments, the buffer size is 120 pkts and the REM parameters are: $\gamma = 0.001$, $\alpha_l = 0.1$, $\phi = 1.1$.

The results are shown in Figure 11. The equilibrium source rate, averaged over all sources, remains steady as propagation delay increases and matches well the theoretical value. The equilibrium link utilization remains above 94% while the equilibrium loss ($< 0.2\%$) and backlog (< 13 packets) remains low.

A.4 **Network size**

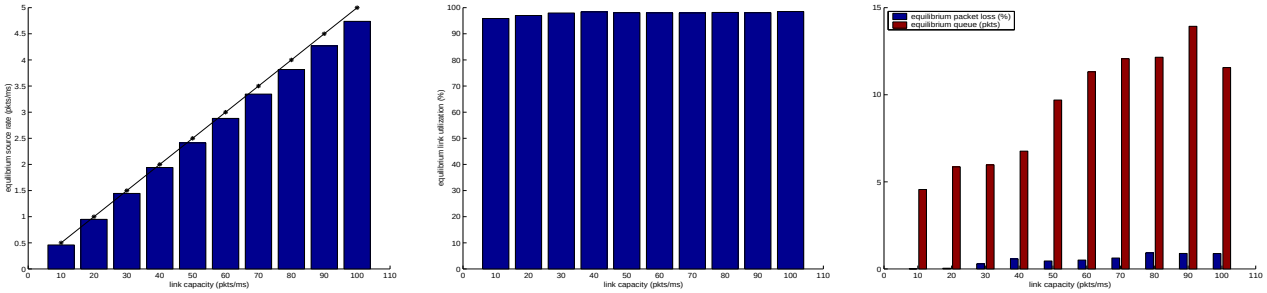
A large network presents two difficulties. First it necessitates a small $\gamma > 0$ in price adjustment, which leads to slower convergence. Second it makes price estimation more difficult, which often leads to wild oscillation and poor utilization. The second difficulty is exposed most sharply in the multilink network of Figure 2(b). When the short connections all have the same utility functions the long connection sees a price that is n times what a short connection sees. It hence sees an end–to–end marking probability that is much larger than that short connections see. Extreme marking probabilities (when n is large) can lead to severe oscillation in the buffer occupancies. The next set of 10 experiments show that a small $\alpha_l (= 0.1)$ reduces the effect of buffer oscillation on prices. This produces smoother price and window processes and a better utilitzation, improving the scalability of REM with network size.

In the simulation, utility functions are $U_s(x_s) = w_s \log x_s$, with w_0 for the long connection set to n times those $w_1 = \cdots = w_n$ for short connections, when there were n links in the network. This is to achieve maxmin fairness, according to which the long connection should receive 50% of bandwidth for all network sizes. We measure both the throughput share of the long connection and link utilization. Throughput share is $x_0^*/(x_0^* + x_i^*)$, $i = 1, \ldots, n$, where x_i^* are the equilibrium rates of connections i, $i = 0, 1, \ldots, n$. Link utilization is $x_0^* + x_i^*$ at link i. The results



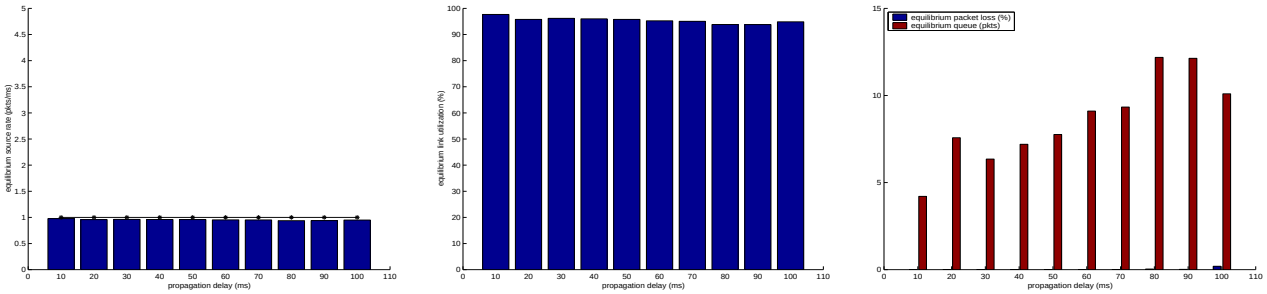
(a) Source rate (b) Link utilization (c) Loss and backlog

Fig. 9. Scalability with traffic load. In (a) each bar represents the measured value and each star '*' represents the theoretical value.



(a) Source rate (b) Link utilization (c) Loss and backlog

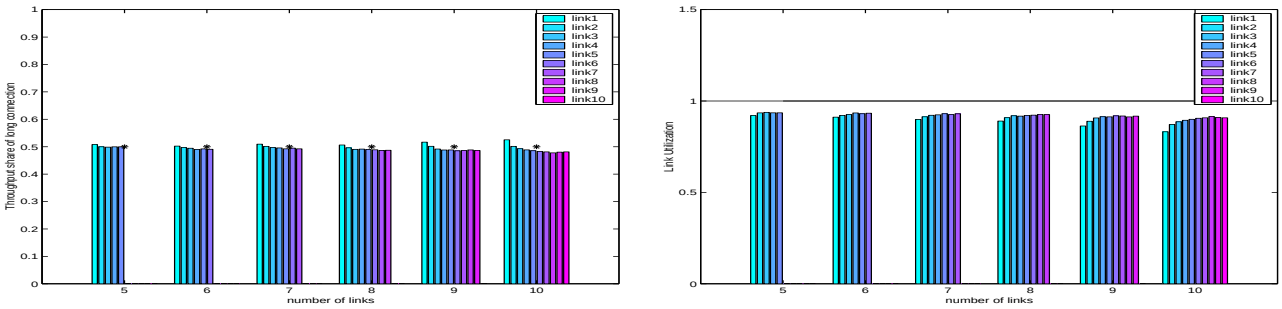
Fig. 10. Scalability with link capacity. In (a) each bar represents the measured value and each star '*' represents the theoretical value.



(a) Source rate (b) Link utilization (c) Loss and backlog

Fig. 11. Scalability with propagation delay. In (a) each bar represents the measured value and each star '*' represents the theoretical value.

are shown in Figure 12 for network sizes above 5. The throughput share matches very well the theoretical value



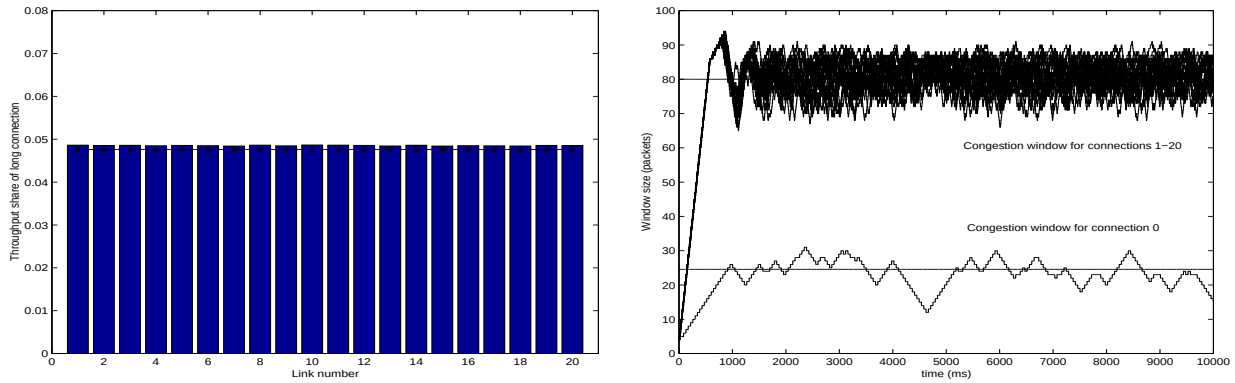
(a) Throughput share of long connection

(b) Link utilization

Fig. 12. Scalability with network size. In (a) each bar represents the measured throughput share at each of the n links and each star '*' represents the theoretical value. In (b) each bar represents the measured utilization at each of the n links and the straight line represents 100% utilization.

and the link utilization is very high. More importantly, the performance remains stable as network size increases.

Figure 13(b) shows the result of another simulation with 20 links. Sources have identical utility parameters to achieve proportional fairness. The throughput shares of the long connection at each of the 20 links are shown in



(a) Throughput share of long connection

(b) Window size

Fig. 13. Scalability with network size. In (a) the straight line shows the theoretical share of 1/21 for the long connection. In (b) the lower curve is the window process for the long connection, and the upper ones are those for the 20 short connections. Marking probability varies over $[0.1972, 0.9876]$.

Figure 13(a). The window process for the 21 connections are shown in Figure 13(b). The performance is very close to expected.

B. **Robustness**

Note that the REM parameters are fixed in each set of the scalability experiments. This demonstrates that REM is robust to parameter setting. The next experiment demonstrates its robustness to error in round trip time estimation.

B.1 **Round trip time estimation**

Round trip time is estimated at sources to translate source rate into window size. A source maintains an exponential weighted average of past round trip times, which is updated on the arrival of every acknowledgement; see the pseudocode in Figure 4.

We use the same setup as that for PC3 and consider both systematic and random error. In the first set of experiments, the estimate at *all* sources was corrupted by constant error of magnitude -1, 0, 1, 2, or 3ms. In the second set of experiments, the estimate at *all* sources was corrupted by additive white Gaussian noise of mean zero and variance one. In the third set of experiments, different sources suffer from different but (randomly chosen) constant errors: S1 suffers no error, S2 -1ms, S3 4ms, and S4 1ms. Since average buffer is small, the average round trip time is close to the round trip propagation delay. The round trip propagation delays of the sources ranges from 7ms to 13ms, and hence the percentage error introduced is very significant.

Table II summarizes the utilization, average backlog, maximum backlog, and average window sizes. As the result indicates REM seems quite robust against errors in round trip time estimation.

Error in RTT	-1	0	1	2	3	random1	random2
Utilization	94.09%	94.33%	95.38%	95.12%	96%	94.94%	95.26%
Avg. backlog (packets)	4.93	4.82	4.90	5.15	5.21	6.07	4.91
Max. backlog (packets)	30	30	27	28	26	37	27
Mean window size							
S1 (21 packets)	19.21	20.54	21.44	21.68	22.84	21.76	19.36
S2 (39 packets)	39.76	38.73	37.93	37.72	37.07	39.67	32.05
S3 (33 packets)	33.06	32.21	32.53	32.28	31.82	31.15	40.68
S4 (27 packets)	26.79	26.66	27.09	27.07	27.56	27.26	27.00
$\frac{\text{Throughput}}{1+\epsilon_s}$							
S1 (k packets)	13.8	12.6	11.5	10.3	9.7	13.1	11.9
S2 (k packets)	14.3	12.9	11.7	10.9	10.0	13.1	11.6
S3 (k packets)	14.3	12.6	11.7	10.7	9.8	12.1	11.7
S4 (k packets)	14.4	12.8	11.7	10.6	9.9	12.9	11.6

TABLE II

ROBUSTNESS TO ROUND TRIP TIME ESTIMATION. WINDOW SIZES IN BRACKETS ARE THEORETICAL EQUILIBRIUM VALUES (UNDER BASIC ALGORITHM). BACKLOG AND WINDOW SIZE DO NOT VARY SIGNIFICANTLY WITH ERROR.

We now offer a heuristic explanation of our simulation results, *assuming* that round trip time (propagation plus queueing delay) is constant. This assumption is reasonable as the average backlog is very small under REM. Let D_s denote the constant round trip time of source s and ϵ_s be the percentage error in its estimation. Then given a path price p^s, source s *appears* to choose its rate x_s according to

$$x_s \;=\; U_s'^{-1}(p^s)\;\frac{(1+\epsilon_s)D_s}{D_s}$$

as opposed to $x_s = U_s'^{-1}(p^s)$. Hence its marginal utility appears to be

$$p^s \;=\; U_s'\left(\frac{x_s}{1+\epsilon_s}\right) \;=:\; \hat{U}_s'(x_s) \tag{15}$$

Since this holds for all x_s integrating (15) implies that source s appears to have a utility function $\hat{U}_s(x_s)$ given by:

$$\hat{U}_s(x_s) \quad = \quad (1+\epsilon_s)\ U_s\left(\frac{x_s}{1+\epsilon_s}\right) \tag{16}$$

Hence round trip time error distorts the source utility function from $U_s(x_s)$ to $\hat{U}_s(x_s)$. (16) allows us to calculate the new equilibrium rates in a general network, given percentage errors. In the distorted utility function $\hat{U}_s$ the rate is reduced by a factor of $1+\epsilon$ but the utility is increased by the same factor. This self–regulating feature provides robustness to error in round trip time estimation. Notice that only the percentage error matters, not the round trip time itself.

For a single–link network, (15) implies that (since all sources see the same path price): for all $r, s \in S$,

$$U_s'\left(\frac{x_s}{1+\epsilon_s}\right) \quad = \quad U_r'\left(\frac{x_r}{1+\epsilon_r}\right) \quad = \quad p^s$$

If U_s are identical and *strictly* increasing, as in our simulations, we must have

$$\frac{x_s}{1+\epsilon_s} \quad = \quad \frac{x_r}{1+\epsilon_r}$$

The last row of Table II shows the equilibrium throughput divided by $1+\epsilon_s$, and as expected, they are similar across sources.

VI. Discussion: parameter setting, comparison with TCP

A. Parameter setting

In this subsection we summarize our experience with parameter setting of REM. The three main parameters are γ and α_l in price adjustment (1), and ϕ in marking probability (2). We emphasize that we do not yet undersand the best way to set these parameters, but will comment on their effect on performance.

The parameter γ must be strictly positive, and usually small. It determines the convergence of REM: $\gamma > 0$ must be small enough for REM to converge, but should not be unnecessarily small so that the rate of convergence is not exceedingly slow [23, Theorems 1 and 2]. We have found that $\gamma = 0.005$ or 0.001 works well.

The parameter α_l must be strictly positive, usually between 0 and 1, and can be different at different links. A large α_l, say, $\alpha_l = 1$, amplifies the effect of backlog on price, and often leads to a small backlog, and hence a low utilization. A small α_l, say, $\alpha_l = 0.1$, improves significantly utilization with only a modest increase in average backlog. The size of α_l thus trades off utilitzation and delay. A small α_l also smoothens the price process by reducing the effect of buffer oscillation on price.[3] Our experience suggests that $\alpha_l = 0.1$ works well.

Among the factors that affect the performance of REM, the most critical is the range of the end–to–end marking probabilities seen by the sources. Extreme probability leads to severe oscillation and poor utilization. The parameter ϕ controls marking probability and must be strictly greater than 1. Ideally it should be chosen so that the end–to–end marking probabilities under nominal traffic condition fall within a range where reasonable price estimation can be made. Call this the *good* range. From our experience (smoothed) REM works well when the probabilities are in $[0.05, 0.99]$. Outside this good range there is severe oscillation in buffer and window processes. The asymmetry of the range (i.e., the right boundary is much closer to 1 than the left boundary is to 0) is due to the fact that, with log utility function, the source rate is sensitive to price when the price, or equivalently the marking probability, is small; see Figure 1 or Section IV-A.

Scalability of REM depends largely on our ability to control the end–to–end marking probability to lie within this range. When we scale up the number of sources or the size of the network, the path prices and hence the end–to–end marking probabilities vary over a wider range. The boundaries of this good range hence determine the number of sources or the size of the network for which REM performs well. Techniques that enlarge the good range improve the scalability of REM.

[3] This can be achieved also by using average, instead of instantaneous, buffer occupancy in price adjustment (1).

B. Comparison with current TCP schemes

We contrast how congestion is measured in TCP Reno [30], TCP Vegas [5], RED [11] and REM. Related marking schemes are discussed in [29], [12], [17].

Network congestion can be measured in different ways. Reno without RED measures congestion with buffer overflow, Vegas measures it with queueing (not including propagation) delay [20], RED measures it with average queue length, and REM measures it with the price vector $p(t) = (p_l(t), l \in L)$. A critical difference among them is the coupling of congestion measure and performance measure, such as loss, delay or queue length, in the first three schemes. We believe that congestion measure should summarize the status of the network such as the available link capacities, the number of sources, their routing and utility functions. This is desirable as they provide to a *new* source the necessary information to decide its rate. The equilibrium value of the congestion measure should depend not on the flow control algorithm used but *solely* on these network conditions. Coupling it with a performance measure creates a dilemma. On the one hand, if the congestion measure reflects network conditions such as the number of sources, then its value steadily increases as more sources activate. This implies that performance such as loss or delay must steadily deteriorate. This is the case with Reno [17, Remark 2] and Vegas [20], where loss, and respectively delay, increases with the number of sources. On the other hand, if the congestion measure, such as the average queue length in RED, should remain steady regardless of the number of sources, then stronger congestion signals must be sent to every source as new sources activate to ensure further reduction in its rate. This necessitates adapting parameters to network conditions as proposed in [8] for RED. A more subtle disadvantage is that the equilibrium value of the congestion measure carries little information about current network conditions, and hence new sources must probe harder. The attempt to decouple congestion measure and network conditions seems fundamental to the difficulty with parameter setting of RED that would work well in different conditions, as observed in [8], [7], [27]. In contrast the equilibrium prices of REM summarize network conditions in a precise sense and are decoupled from performance measure; see Section II. This is the underlying reason for the robustness of REM to parameter setting. Indeed the prices steadily increase as new sources activate while the backlog is made to always converge to a neighborhood of zero. High utilitzation is achieved not through maintaining a large backlog, but through the precise congestion information sources obtain from marking to set their rates.

In summary, if congestion measure is coupled with performance measure, then 'congestion' necessarily means 'bad performance' such as large loss or delay. If they are decoupled, as in REM, then 'congestion' simply means that 'demand for exceeds supply of' network resources. This curbs demand but maintains good performance, such as low delay and loss.

VII. Interworking with Reno

The link algorithm of REM, (1–2), can also interwork with existing source algorithms. Its unique advantages are that it provides each source with a congestion measure that is aggregated over its path, and that it tends to clear the buffer, leading to low loss and delay. In this section we present preliminary simulation results to compare the performance of Reno (with DropTail), Reno/RED and Reno/REM.

All simulations have been done using the *ns-2* simulator for the single link network of Figure 2(a) with n sources. All sources have the same round trip propagation delay of 120ms, with one-way propagation delay of 30ms on each (access or shared) link. The size of each packet is 1KB. At the router a buffer with capacity of 100 packets is used. ECN is set so that packets are dropped only when buffer overflows. Packets are served in FIFO order and are marked with a probability determined by the link algorithm (REM or RED). Note that the (equilibrium) marking probability is determined by the equilibrium source rates and their round trip times and is independent of the link algorithm [22]. REM exhibits a much smoother marking probability than RED; see a more comprehensive comparison in [2].

Eleven experiments have been run with $n = 5, 10, 20, \ldots, 100$ sources each. REM parameters are: $\gamma = 0.01$,

$\phi = 1.001$, $\alpha = 0.1$. In addition we have substituted $b_l(t)$ in the price update (1) by $b_l(t) - b_0$ with $b_0 = 20$ packets. This has the effect of maintaining an equilibrium queue length of around $b_0 = 20$ packets while increasing the link utilization during transient. RED parameters are: min thresh = 20KB, maximum thresh = 80KB and $wq = 0.002$. These parameter values are used in all the eleven experiments. Each experiment runs for 30s. Equilibrium link utilization, queue length and loss are measured over the last 15s.

The results are shown in Figures 14, 15 and 16. All three schemes achieve high link utilization with Reno without

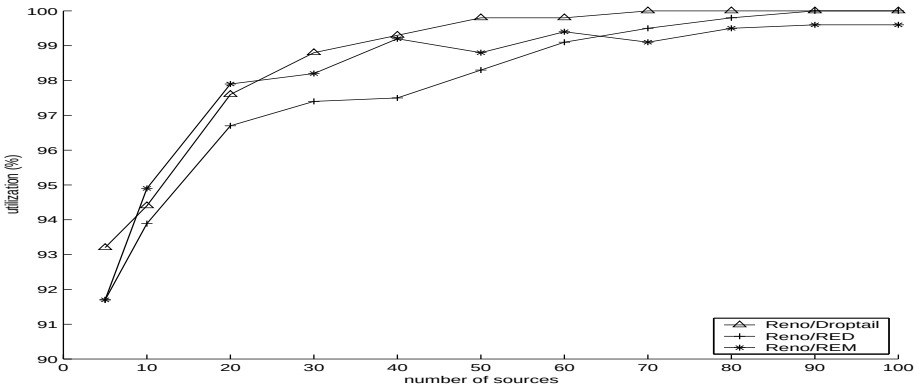


Fig. 14. Link utilization

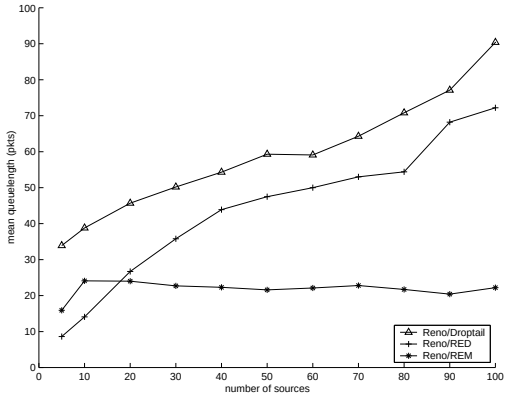


Fig. 15. Average backlog

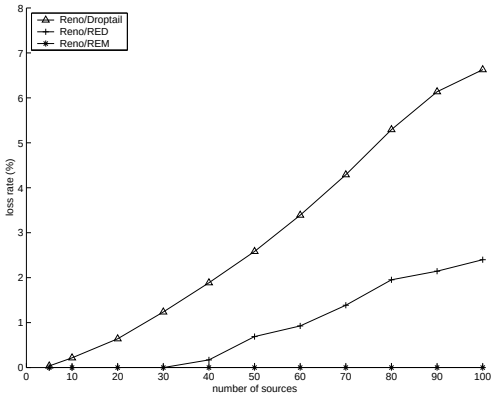


Fig. 16. Loss

marking having the highest. The average queue length and loss remain very low with Reno/REM as load increases, while they steadily increase with Reno and with Reno/RED.

VIII. **Conclusion**

We have presented a Random Exponential Marking algorithm for flow control as a practical implementation of the basic algorithm of [23]. The algorithm is summarized in Section I-B and a pseudocode implementation is given in Section III. Extensive simulations indicate that it is stable, fair, scalable, and robust; it achieves high utilization with negligible loss or queueing delay. Its key features are summarized in Section I-C. REM owes its robustness and good performance fundamentally to the way it meausres congestion, as discussed in Section VI.

The preliminary results on Reno/REM and the more comprehensive simulations in [2] suggest that it would be advantageous to deploy REM in routers for active queue management. It is simple, scalable, and achieves high utilization with negligible loss or delay. Moreover, for sources that are not ECN-capable, routers can just drop, as

opposed to mark, their packets according the REM algorithm. This does not require *any* modification to Reno. REM can also be applied in a (private sub-) network to control the aggregate. Then new source algorithms, e.g., TCP Vegas, as well as the link algorithm, can be implemented to maximize performance.

We comment on two limitation of this work. First REM prescribes a way to control rate–adaptive flows to achieve social optimality. It does not however itself provide incentive for sources to cooperate, a critical but open problem. Congestion pricing is a possibility that aligns social and individual optimality. Second even though REM seems robust to parameter setting, a potential difficulty is that ϕ must be chosen in a way that maintains marking probabilities to within a good range. Moreover this is a constant that must be fixed and known globally. A critical future work is to investigate ways to alleviate or get around this difficulty.

Acknowledgements: We gratefully acknowledge very helpful discussions with Sally Floyd, Frank Kelly, David Lapsley, and K. K. Ramakrishnan.

References

[1] Sanjeewa Athuraliya, David Lapsley, and Steven Low. An Enhanced Random Early Marking Algorithm for Internet Flow Control. In *Proceedings of IEEE Infocom*, March 2000.

[2] Sanjeewa Athuraliya, Victor H. Li, and Steven H. Low. Simulation comparison of RED and REM. Submitted for publication, May 2000.

[3] Yair Bartal, J. Byers, and D. Raz. Global optimization using local information with applications to flow control. In *STOC*, October 1997.

[4] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[5] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995.

[6] Costas Courcoubetis, Vasilios A. Siris, and George D. Stamoulis. Integration of pricing and flow control for ABR services in ATM networks. *Proceedings of Globecom'96*, November 1996.

[7] W. Feng, D. Kandlur, D. Saha, and K. Shin. BLUE: a new class of active queue management algorithms. Technical report, University of Michigan, Michigan, USA, 1999. UM CSE–TR–387–99.

[8] W. Feng, D. Kandlur, D. Saha, and K. Shin. A self–configuring RED gateway. In *Proceedings of INFOCOM'99*, March 1999.

[9] S. Floyd. Connections with multiple congested gateways in packet–switched networks, Part I: one–way traffic. *Computer Communications Review*, 21(5), October 1991.

[10] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5), October 1994.

[11] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, August 1993.

[12] R. J. Gibbens and F. P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, 35, 1999.

[13] Jamal Golestani and Supratik Bhattacharyya. End-to-end congestion control for the Internet: A global optimization framework. In *Proceedings of International Conf. on Network Protocols (ICNP)*, October 1998.

[14] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997. http://www.statslab.cam.ac.uk/f̃rank/elastic.html.

[15] Frank Kelly. Mathematical modelling of the Internet. Preprint, 1999.

[16] Frank P. Kelly, Aman Maulloo, and David Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of Operations Research Society*, 49(3):237–252, March 1998.

[17] Srisankar Kunniyur and R. Srikant. End–to–end congestion control schemes: utility functions, random losses and ECN marks. In *Proceedings of IEEE Infocom*, March 2000.

[18] David E. Lapsley and Steven H. Low. An optimization approach to ABR control. In *Proceedings of the ICC*, June 1998.

[19] David E. Lapsley and Steven H. Low. Random Early Marking for Internet Congestion Control. In *Proceedings of IEEE Globecom'99*, December 1999.

[20] Steven Low, Larry Peterson, and Limin Wang. Understanding Vegas: theory and practice. Submitted for publication, http://www.ee.mu.oz.au/staff/slow/research/, February 2000.

[21] Steven H. Low. Optimization flow control with on-line measurement. In *Proceedings of the ITC*, volume 16, June 1999.

[22] Steven H. Low. Flow control through duality. Submitted for publication, March 2000.

[23] Steven H. Low and David E. Lapsley. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999. http://www.ee.mu.oz.au/staff/slow/research/.

[24] David G. Luenberger. *Linear and Nonlinear Programming, 2nd Ed.* Addison-Wesley Publishing Company, 1984.

[25] L. Massoulie and J. Roberts. Bandwidth sharing: objectives and algorithms. In *Infocom'99*, March 1999. Available at http://www.dmi.ens.fr/%7Emistral/tcpworkshop.html.

[26] Jeonghoon Mo and Jean Walrand. Fair end–to–end window–based congestion control. Preprint, 1999.

[27] T. J. Ott, T. V. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proceedings of IEEE Infocom'99*, March 1999.

[28] K. K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. Internet draft draft-kksjf-ecn-01.txt, July 1998.

[29] K. K. Ramakrishnan and Ran Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems*, 8(2):158–181, May 1990.

[30] W. Stevens. *TCP/IP illustrated: the protocols*, volume 1. Addison–Wesley, 1999. 15th printing.