Optimization Based Congestion Control for the Internet *

Steven H. Low
Dept of Electrical & Electronic Engineering
University of Melbourne, Australia
slow@ee.mu.oz.au

August 15, 1999

Abstract

Optimization based flow control is motivated by a welfare maximization problem and derives its control mechanism as a distributed algorithm to solve it. Different proposals differ in their definition of welfare or solution approach and lead to different link and source algorithms. We provide a brief and coherent description of four such proposals and show how they can be implemented on the Internet using binary or implicit feedback.

1 Introduction

Congestion control is a set of distributed algorithms to share network resources among competing users.[1] They adapt to fluctuations in the capacity of and the demand for these resources. They often consist of two components: *link algorithm* executed by network devices such as routers or switches, and *source algorithm* executed by host computers or edge devices. Link algorithm detects congestion and feeds back information to users, and in response, source algorithm adjusts the rate at which user traffics are injected into the network. The basic design issue is what to feed back (link algorithm) and how to react (source algorithm) and the goal is to achieve stability, optimality, and fairness. As we will see below, several important schemes on the Internet do not have link algorithm and require the source algorithm to infer network congestion from local observation such as round trip time (the time from sending a packet to receiving its acknowledgement) and packet loss.

There is a tremendous literature on congestion control and it is impossible to provide a complete survey of major contributions in the limited space. The purpose of this paper is to give a brief but coherent description of a recent

*Submitted to IEEE Network as a tutorial for a general audience.
[1]We use 'congestion control' and 'flow control' interchangeably.

optimization approach to flow control. Different proposals of this approach all motivate flow control by a certain welfare maximization problem and derive flow control mechanisms as distributed algorithms to solve it. They differ in their definition of welfare function or solution approach, and result in different link and source algorithms. They are described in Section 3.

The value of the optimization framework is twofold. First, though it may not be possible, nor critical, that optimality is exactly attained in a real network, the optimization framework offers a means to explicitly steer the network *as a whole* towards a desirable operating point. Second, it is useful to treat practical flow control schemes simply as distributed and asynchronous implementations of a certain optimization algorithm. The optimization model then makes possible a systematic method to design and refine these schemes, where modifications to a flow control mechanism are guided by modifications to the optimization algorithm. See Section 3.3 for examples.

The TCP/IP (Transport Control Protocol/Internet Protocol) currently provides no facility for network devices to explicitly feed back congestion information to users. There is however a major proposal to use a single bit in the IP header for such purpose [4, 16]. This imposes a constraint on the link algorithm as well as the source algorithm. We will explain how the optimization based algorithms observe this important constraint. To provide the context, we first describe in the next section three original proposals for TCP flow control.

2 Current methods

In this section we will briefly describe TCP tahoe and reno [8], TCP vegas [2], and RED (Random Early Detection) [5]. The first two schemes use implicit feedback and has no link algorithm, while the third scheme can be implemented using a single–bit feedback. TCP also provides other end–to–end services such as error recovery and round trip time estimation (some of which affect congestion control) but we will limit our attention to the congestion control aspect. Due to space limitation we will not cover the many enhancements to these schemes.

2.1 TCP tahoe and reno

The basic idea of TCP tahoe and reno is for a source to gently probe the network for spare capacity by linearly increasing its rate, and exponentially reducing its rate when congestion is detected. Congestion is detected when the source detects a packet loss. TCP controls its sending rate via window size, the maximum number of outstanding packets: a large the window corresponds to a high sending rate. When a connection starts, it is prudent to start with a small window (size =1) and the source increases its window by 1 every time it receives an acknowledgement. This doubles the window rapidly to a value close to matching the available capacity. This is the *slow–start* phase. Then the source enters the *congestion avoidance* phase where it increases its window by the reciprocal of the current window size every time it receives an acknowledgement.

This has the effect of increasing the window by 1 in each round trip time, and is referred to as additive or linear increase. The transition from the slow–start phase to the congestion avoidance phase is made when the window exceeds a threshold. This threshold is meant to indicate the available capacity in the network and is adjusted each time a loss is detected. The source algorithm is summarized in Figure 1. W is the current window size and $\overline{W}$ is the threshold on W that determines whether the source is in the slow–start phase or the congestion avoidance phase.

```
For every ack received
{
   if W < W̄ then   (slow–start phase)
        W ⟵ W + 1;
   else   (congestion avoidance phase)
        W ⟵ W + 1/W;
}
```

(a) Additive increase

```
For every loss detected
{
   W̄ ⟵ W/2;
   W ⟵ 1;
}
```

(b) Multiplicative decrease

Figure 1: TCP Tahoe

The algorithm in Figure 1 was first proposed in [8] and implemented in the tahoe version of TCP. Two key refinements were subsequently implemented in TCP reno to detect and recover from loss more efficiently. TCP tahoe detects a packet loss, and retransmits the packet, when the associated timer expires. Due to the coarse clock resolution (500 ms) used for retransmit timeout, it can take four times longer than necessary to detect a loss [2]. The first refinement, called Fast Retransmit, detects loss and starts retransmission either when a timeout occurs or when the source receives three duplicate acknowledgements. This significantly shortens the delay in detecting a loss and reacting to congestion. On detecting a loss, TCP tahoe retransmits the lost packet and immediately resets

the window to 1 (Figure 1(b)). This means that a new packet can be transmitted only after a round trip time when the acknowledgement for the retransmitted packet arrives at the source. Moreover the 'pipe' (the path between the source and the receiver when they are the only TCP connection) is cleared when the retransmitted packet reaches the receiver, and some of the routers in the path become idle during this period, resulting in loss of efficiency. The second refinement in TCP reno, called Fast Recovery, allows the window W to temporarily grow *exponentially* beyond $\overline{W}$ in order to keep the pipe filled. Specifically, suppose the loss of packet l is detected. The source sets the threshold to half of the current window, $\overline{W} \longleftarrow W/2$, and sets the window $W \longleftarrow \overline{W} + 3$. On each acknowledgement, the source increases the window W by 1 so that W grows exponentially. Note that when W reaches $2\overline{W} + 1$ the source can transmit new packets again. Eventually the (nonduplicate) acknowledgement for the retransmitted packet l arrives. This ends the Fast Recovery period. The source then resets the window $W \longleftarrow \overline{W}$ and resumes the normal congestion avoidance mechanism of tahoe. With the Fast Retransmit and Fast Recovery mechanisms, TCP reno detects and recovers from congestion more quickly than TCP tahoe. Moreover the slow–start phase is entered only when the connection first starts. TCP reno halves its window and enters congestion avoidance phase directly after Fast Recovery from a loss. This signifcantly improves the throughput.

2.2 TCP vegas

TCP vegas [2] improves upon TCP tahoe and reno through three techniques. The first is a new retransmission mechanism where timeout is checked on receiving the first duplicate acknowledgement, rather than waiting for the third duplicate acknowledgement (as reno would), and results in a more timely detection of loss. The second technique is a more prudent way to grow the window size during the initial use of slow–start when a connection starts up, and results in less losses.

Reno discovers the available bandwidth in the network by continually increasing its window size until it congests the network and packets are lost. It then drastically reduces its window size, and the cycle repeats. The third technique of vegas is a new congestion avoidance mechanism that corrects such oscillatory behavior. The basic idea is to have a source estimate the number of its own packets buffered in the path, and try to keep this number between α (typically 1) and β (typically 3) by adjusting its window size. The window size is increased or decreased linearly in the next round trip time according as the current estimate is less than α or greater than β. Otherwise the window size is unchanged. The rationale behind this is to maintain a small number of packets in the pipe to take advantage of extra capacity when it becomes available. Specifically let d denote the round trip propagation delay (assuming processing time is negligible). Let D be the round trip time, i.e., d plus queueing delay. Let W denote the current window size. Then W/D represents the current source rate and W/d represents the maximum source rate achievable. As we will see below, the ratio $d/D(\leq 1)$ is related to the number of packets buffered in the

pipe. Vegas estimates the current value of D once every round trip time, and sets d to the minimum D observed so far. The algorithm is shown in Figure 2.

```
For every round trip time
{
    if W(1 − d/D) < α then W ⟵ W + 1;
    if W(1 − d/D) > β then W ⟵ W − 1;
}
```

Figure 2: TCP Vegas

To see why this algorithm controls the number of packets queued in the path, consider the equilibrium situation where there are b bits in the path belonging to a given vegas source. Then

$$D \quad = \quad d + b/c$$

where c is the capacity available for the source. The source rate equals the capacity c in equilibrium and hence the window size is given by $W = Dc$. The difference in the vegas algorithm is then the equilibrium buffer occupancy b:

$$W\,(1 - d/D) \quad = \quad Dc \cdot \frac{b}{Dc} \quad = \quad b$$

Hence controlling the difference to within α and β maintains a small number of packets buffered in the pipe. This makes efficient use of fluctuating capacity without slowly pushing the network into congestion, as tahoe and reno tend to do. This can be shown to achieve proportional fairness; see Section 3.5.

2.3 RED

TCP tahoe and reno detects congestion only after the fact when buffer has already overflown and packets lost. RED (Random Early Detection) [5] attempts to avoid congestion by providing warning to sources when congestion starts to develop. A major departure from the previous schemes is that RED employs a link algorithm to detect congestion and provide a single–bit feedback to sources.

Congestion at a link is measured by queue length. With RED, a link maintains an (autoregressive–moving–) average queue length. It marks a packet with a probability that increases with the average queue length. When the average queue length is less than a minimum threshold, no packets are marked. When it exceeds a maximum threshold, all packets are marked. When it is in between, a packet is marked with a probability that is a convex increasing function of the average queue length.

Specifically, RED calculates a probability p_b that grows linearly from 0, when the average queue length b equals the minimum threshold $\underline{b}$, to a maximum value

$\overline{p} << 1$, when the average queue length b equals the maximum threshold $\overline{b}$:

$$p_b = \overline{p}\frac{b - \underline{b}}{\overline{b} - \underline{b}}$$

The marking probability p_a is then given by

$$p_a = \frac{p_b}{1 - np_b} \tag{1}$$

where n is the number of unmarked packets since the last marked packet. The rationale behind (1) is to make the interval between marked packets uniformly distributed. The marking probability p_a is thus convex increasing in the average queue length b, as shown in Figure 3(a).

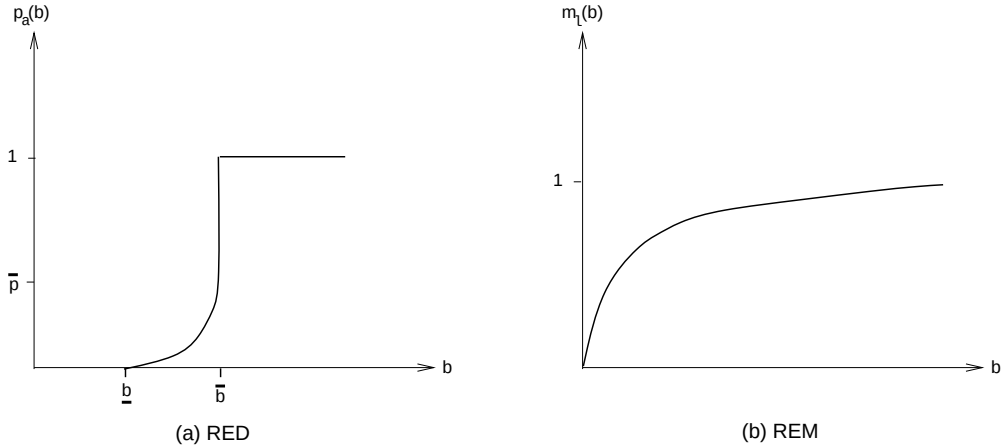

Figure 3: Link marking probability as a function of average queue length

RED marks a packet by setting the proposed ECN (Explicit Congestion Notification) bit in IP header [4, 16]. The mark is reflected back to the source in the ECN bit of the acknowledgement.

RED in [5] assumes that sources treat a mark like a packet loss and react according to TCP tahoe or reno. It improves upon tahoe and reno by providing early warning of congestion. With RED the queue length does not significantly exceed the maximum threshold, whereas without it, the queue often oscillates between overflowing and underflowing.

Probabilistic marking has two added advantages. First, it avoids global synchronization where all sources detect congestion and reduce their rates at the same time. Second, under first–in–first–out service discipline, the fraction of marked packets of a source is roughly proportional to the fraction of capacity the source enjoys, and hence by monitoring the most frequently marked sources, it is easy to identify misbehaving sources that monopolize resources to the detriment of others.

An alternative to marking is to probabilistically drop packets to warn sources of incipient congestion. This has the advantage of not requiring explicit feedback but has significantly worse performance if the dropping probability is high. If the dropping probability is low, or packets are dropped only when buffer overflows, then the number of dropped packets per unit flow can severely underestimate the true congestion costs at large resources [6]. In other words the feedback signal sources receive need to be much more timely and stronger than dropping provides.

3 Optimization based methods

TCP tahoe, reno and vegas provide methods for sources to discover available bandwidth in the network. RED provides early warning of congestion to the sources. These clever algorithms are all local in nature, e.g., there is no rigorous coordination between the congestion signal generated by the link algorithm and the reaction by the source algorithm. Yet, these algorithms seem stable when work together over a large network. The dynamics of such a network are however difficult to understand.

In contrast, the methods reviewed in this section are all derived from a global optimization framework. The behavior of the network as a whole is thus under explicit control – to track a global optimality. Unlike previous methods the link algorithm and the source algorithm must be designed *jointly*. Indeed we can think of the links and sources as processors in a distributed computation system that are collaborating in maximizing a welfare function. Each processor executes a simple algorithm, exchanges results, and repeats the cycle. In this context a link calculates a congestion signal and a source selects a transmission rate in response.

The communication between these processors (links and sources) is constrained by the current and pending Internet standards. Of the four algorithms we review, the first two assume the availability of a single bit for congestion feedback, like RED does, and the last two use only implicit feedback, as TCP tahoe, reno and vegas do. All four algorithms motivate flow control by welfare maximization, but differ in their solution approach. We first describe a general setup, and then explain how each solves the general optimization problem.

3.1 General setup

A key premise of optimization based flow control is that sources with different valuation of bandwidth should react differently to congestion. Faced with the same congestion, a source with a higher valuation should send at a higher rate than one with a lower valuation. This is modeled by different source utility functions; see [3] for empirical measurement of utility functions (or equivalently, demand curves) for Internet access.

Consider a network that consists of a set $L = \{1, \ldots, L\}$ of unidirectional links of capacities c_l, $l \in L$. The network is shared by a set $S = \{1, \ldots, S\}$ of

sources. Source s is characterized by two parameters $(L(s), U_s)$ where the path $L(s) \subseteq L$ is a set of links that source s uses, and $U_s : \Re_+ \to \Re$ is a utility function. Source s attains a utility $U_s(x_s)$ when it transmits at rate x_s. We assume $U_s(\cdot)$ is increasing and strictly concave 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. Note that $l \in L(s)$ if and only if $s \in S(l)$.

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

$$\max_{x_s \geq 0} \quad \sum_s U_s(x_s) \tag{2}$$

$$\text{subject to} \quad \sum_{s \in S(l)} x_s \ \leq \ c_l, \quad l = 1, \ldots, L. \tag{3}$$

The constraint (3) says that the total source rate at any link l does not exceed the capacity. Since all utility functions are strictly concave, a unique optimal rate vector x exists. Solving this problem centrally is impractical as it would require not only the knowledge of all utility functions, but worse still, complex coordination among potentially all sources due to coupling of sources through shared links. As we will see, flow control mechanisms are derived as distributed algorithms to compute the optimal rate vector.

Though the setup is static where the link capacities c_l, the set S of sources and their utility functions U_s remain fixed, the source and link algorithms generalize directly to the case when these quantities are time-varying; see Section 3.3 below. They have the important virtue of not requiring to be restarted when network condition changes. Provided the change in link capacities and sources is slow relative to the convergence of the algorithms, the algorithms track the moving optimal rates.

3.2 Proportional fairness

The approach taken in [9, 10, 6] decomposes the welfare maximization (2–3) into a user subproblem and a network subproblem. Given the price p^s for a unit of bandwidth, the user subproblem for source s is to choose a willingness–to–pay w_s (which can purchase an amount $x_s = w_s/p^s$ of bandwidth) in order to maximize its benefit:

$$\text{USER}_s(U_s; p^s): \qquad \max_{w_s \geq 0} \ U_s\left(\frac{w_s}{p^s}\right) - w_s$$

Given users' willingness–to–pay vector $(w_s, s \in S)$, the network subproblem is to choose source rates $(x_s, s \in S)$ in order to maximize $\sum_s w_s \log x_s$:

$$\text{NETWORK}(A, c; w): \qquad \max_{x_s \geq 0} \ \sum_s w_s \log x_s$$

$$\text{subject to} \qquad Ax \ \leq \ c$$

where $A_{ls} = 1$ if $l \in L(s)$ and zero otherwise is the routing matrix, $c = (c_l, l \in L)$ is the vector of link capacities, and the constraint is simply (3) in matrix notation. It is shown in [9] that there exist prices $(p^s, s \in S)$, source rates $x = (x_s, s \in S)$ and willingness–to–pay $(w_s, s \in S)$ with $w_s = p^s x_s$ such that w_s solves the subproblem $\text{USER}_s(U_s; p^s)$, and x solves the subproblem $\text{NETWORK}(A, c; w)$. Moreover x solves the welfare maximization (2–3).

A rate vector x^* is *proportionally fair* if it is feasible ($x^* \geq 0$ and $Ax^* \leq c$) and if for any other feasible vector x, the aggregate of proportional changes is nonpositive:

$$\sum_s \frac{x_s - x_s^*}{x_s^*} \leq 0$$

A vector x^* is such that the *rates per unit charge* are proportionally fair if x^* is feasible and if for any other feasible vector x

$$\sum_s w_s \frac{x_s - x_s^*}{x_s^*} \leq 0 \tag{4}$$

It is shown in [9] that x^* solves the subproblem $\text{NETWORK}(A, c; w)$ if and only if the rates per unit charge are proportionally fair. Major effort in [10] is then to solve the subproblem $\text{NETWORK}(A, c; w)$, or equivalently, its dual $\text{DUAL}(A, c; w)$, given users' willingness–to–pay w, to obtain a rate vector $x(w)$ that satisfies (4). User adaptation can then update w, possibly at a slower time scale, such that $x(w)$ maximizes the welfare (2–3).

To this end they propose the following primal algorithm

$$\frac{d}{dt} x_s(t) = \gamma \left(w_s - x_s(t) \sum_{l \in L(s)} p_l(t) \right) \tag{5}$$

$$p_l(t) = g_l \left(\sum_{s \in S(l)} x_s(t) \right) \tag{6}$$

to solve a relaxation of the subproblem $\text{NETWORK}(A, c; w)$, and the following dual algorithm to solve a relaxation of $\text{DUAL}(A, c; w)$

$$\frac{d}{dt} p_l(t) = \gamma \left(\sum_{s \in S(l)} x_s(t) - q_l(p_l(t)) \right) \tag{7}$$

$$x_s(t) = \frac{w_s}{\sum_{l \in L(s)} p_l(t)} \tag{8}$$

In the primal algorithm (5–6), $p_l(t)$ is a congestion signal (price) generated by link l as a function of aggregate source rate at link l. The price faced by source s is the sum of link prices in the path, $p^s(t) = \sum_{l \in L(s)} p_l(t)$. The rate adjustment (5) adopts multiplicative decrease and additive increase as TCP tahoe and reno do. The nonnegativity constraint on the rates and prices are

relaxed in the model (5) and (7). This simplification allows an elegant proof in [10] via Lyapunov argument that the primal algorithm (5–6) and the dual algorithm (7–8) are globally stable. Moreover the rate vector $x(t)$ converges to one that solves a relaxation of NETWORK$(A, c; w)$ and the charge $x_s(t)p^s(t)$ converges to the willingness–to–pay w_s.

To apply the primal algorithm (5–6) to IP networks, [6] proposes binary–feedback schemes where marks convey to a source the charge $x_s(t)p^s(t)$ and the source adjusts its rate to equalize the charge with its willingness–to–pay w_s. The charge $p^s(t)$ per unit flow is taken to be the *shadow price* of a single bottleneck link, i.e., the marginal increment in expected cost (defined below) at the link for a marginal increment in load. The marking scheme is derived from the following single link model. Suppose time is slotted and consider a link that can serve N packets per slot, with any excess lost. Suppose the link is shared by S sources. In slot t they generate packets whose numbers are independent Poisson random variables $X_1(t), X_2(t), \ldots, X_S(t)$ with means $x_1(t), x_2(t), \ldots, x_S(t)$. The total load in slot t is thus a Poisson random variable $Y(t) = \sum_s X_s(t)$ with mean $y(t) = \sum_s x_s(t)$. Define the cost to be the number of packets loss. Then the expected cost $C(y(t))$ per slot is given by

$$C(y(t)) \quad = \quad E(Y(t) - N)^+ \quad = \quad \sum_{n \geq N} (n - N) e^{-y(t)} \frac{y^n(t)}{n!}$$

Let $p(t) = \frac{d}{dy} C(y(t))$ be the shadow price. Suppose now that whenever the number $Y(t)$ of packets arriving in slot t exceeds N, *all* these packets are marked. Then the expected number of marks per slot placed on packets of source s can be shown to be proportional to the shadow price:

$$E(X_s(t) 1\{Y(t) > N\}) \quad = \quad x_s(t) p(t)$$

where $1\{E\} = 1$ if event E occurs and zero otherwise. Hence by marking every packet when the link is overloaded and treating each mark as a charge per unit time, every source receives a charge per unit bandwidth that is precisely equal to the shadow price $p(t)$ at the bottleneck link.

The insight from the above bufferless model is applied to a buffered model, where all packets arriving between the start of a busy period and the loss, within the same busy period, of a packet should be marked, because only an additional packet within this time interval will increase the number of lost packets. This marking strategy however is noncausal and hence not implementable. An approximate mechanism is to mark all packets in the queue at the time of a packet loss, and then mark a sufficient number of immediately subsequent packets to ensure that, in total, the correct number of packets are marked.

3.3 REM

Instead of decomposing the problem (2–3) into user and network subproblems, the approach in [11, 14, 12] directly considers the dual of (2–3) (not to be

confused with the subproblem DUAL$(A, c; w)$ which is the dual of the network subproblem):

$$\min_{p \geq 0} \quad D(p) \quad = \quad \sum_s B_s(p^s) + \sum_l p_l c_l \tag{9}$$

where

$$B_s(p^s) \quad = \quad \max_{x_s \geq 0} \; U_s(x_s) - x_s p^s \tag{10}$$

$$p^s \quad = \quad \sum_{l \in L(s)} p_l \tag{11}$$

The first term of the dual objective function $D(p)$ is decomposed into S separable subproblems (10–11). If we interpret p_l as the price per unit bandwidth at link l then p^s is the total price per unit bandwidth for all links in the path of s. Hence $x_s p^s$ represents the bandwidth cost to source s when it transmits at rate x_s, and $B_s(p^s)$ represents the maximum benefit s can achieve at the given price p^s. This scalar p^s summarizes all the congestion information source s needs to know.

For each price vector p, a unique maximizer of (10), denoted by $x_s(p)$, exists since U_s is strictly concave. In general $(x_s(p), s \in S)$ may not be primal optimal, but by duality theory, there exists a dual optimal price $p^* \geq 0$ such that $(x_s(p^*), s \in S)$ is indeed primal optimal. Hence we focus on solving the dual problem (9). Given p^*, individual sources s can solve (10) *separately without the need to coordinate with other sources.* In a sense p^* serves as a coordination signal that aligns individual optimality of (10) with social optimality of (2).

We solve the dual problem using gradient projection method where, in each iteration, link prices are adjusted in opposite direction to the gradient $\nabla D(p)$. This takes the following form of flow control:

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

$$x_s(t+1) \quad = \quad U_s'^{-1}(p^s(t)) \tag{13}$$

where $\gamma > 0$ is a stepsize, and $[z]^+ = \max\{z, 0\}$. Here, $x^l(t) = \sum_{s \in S(l)} x_s(t)$ is the aggregate source rate at link l at time t, and $p^s(t) = \sum_{l \in L(s)} p_l(t)$ is the price per unit bandwidth in the path of source s. The price adjustment (12) follows the law of supply and demand: if the demand $x^l(t)$ for bandwidth exceeds the supply c_l, raise the price; otherwise reduce it. The rate adjustment (13) solves (10) with the current price $p^s(t)$ in place of p^s. Indeed the right hand side of (13) is the demand function in economics: a higher price (congestion) induces a lower rate.

As noted above, though the algorithms are derived from a static model, they generalize directly to the case of slowly time-varying environment. Each source that comes on board executes the same source algorithm with the time invariant utility function $U_s(\cdot)$ replaced by the current utility $U_s(\cdot; t)$ at time t. Similarly each link executes the same link algorithm, with $c_l(t)$ being the current link

capacity and $x^l(t)$ the source rate aggregated over the set $S(l;t)$ of currently active sources through link l.

The algorithm was first proposed in [11]. In [14] we prove the global convergence of both the synchronous and asynchronous versions of the algorithm (12–13) in a static network, and present experimental measurements that demonstrate their convergence when network condition changes slowly.

The basic algorithm (12–13) is not implementable on the Internet because of the communication requirements between links and sources. To update its price, a link requires the aggregate *source* rate, which is generally different from the aggregate *offered* rate at the link (unless it is the first link in the path) and hence must be communicated from the sources using that link. To adjust its rate, a source requires a scalar feedback of price from the network. We now explain how these communication requirements can be greatly simplified.

First a link can use the offered rate as an estimate of the source rate. This is equivalent to estimating the gradient of the dual objective function (9) when carrying out the gradient projection algorithm. Moreover with this approximation, a link can simply set its price to a fraction of its (average) buffer occupancy. We prove in [13] that the estimation error tends to zero and the algorithm converges to yield the optimal rate vector. This eliminates the need for explicit communication from sources to links.

In the reversed direction we achieve the communication from links to sources using a single bit. Each link maintains a price either by setting it to a fraction of buffer occupancy or by updating it using the aggregate offered rate. When a packet arrives at link l, if it is already marked, the mark is not modified. Otherwise link l marks the packet with a probability $m_l(t)$ that is exponential in the current price, independently of all other packets:

$$m_l(t) \quad = \quad 1 - 2^{-p_l(t)}$$

The marking probability m_l as a function of buffer occupancy (when price is set to a fraction of buffer occupancy) is illustrated in Figure 3(b). For a packet of source s, the end–to–end marking probability is:

$$m^s(t) \quad = \quad 1 - \Pi_{l \in L(s)}(1 - m_l(t)) \;=\; 1 - 2^{-p^s(t)}$$

i.e., it is exponential to the bandwidth price source s needs for its rate adjustment (see (13)). Hence source s estimates $m^s(t)$ by the fraction $\hat{m}^s(t)$ of its packets marked in period t and then estimate the price by:

$$\hat{p}^s(t) \quad = \quad -\log_2(1 - \hat{m}^s(t))$$

It then uses the estiamte $\hat{p}^s(t)$ in place of $p^s(t)$ in its rate adjustment (13). This is the REM (Random Early Marking) scheme and can be implemented with binary feedback like RED. Performance of this algorithm is presented in [12].

As noted in Section 1 the optimization model provides a framework to systematically refine practical flow control schemes. For instance, it is well known that Newton algorithm has much faster convergence than gradient projection algorithm. By replacing the gradient projection algorithm presented here by the

Newton algorithm we have derived in [1] a practical Newton–like flow control scheme that can be proved to maintain optimality, has the same communication requirement as the basic scheme (12–13) but enjoys a much better convergence property. We have also applied pole placement technique in linear control to the model here to stabilize its transient in the face of large feedback delays. This has led to a more robust REM.

We now comment on the relationship between this work and that in [9, 10, 6]. The work in [9, 10, 6] decomposes the welfare maximization (2–3) into user and network subproblems, where a user specifies a willingness–to–pay and the network searches for a rate vector such that the rates per unit charge are proportionally fair. This leads to the primal algorithm (5–6) to solve the network subproblem, and the dual algorithm (7–8) to solve the dual of the network subproblem. The work in [11, 14, 12] in contrast solves the welfare maximization (2–3) by minimizing its dual (9), using gradient projection algorithm. In the special case where all utility functions are $U_s(x_s) = w_s \log x_s$, the welfare maximization (2–3) and its dual (9) are identical to NETWORK$(A, c; w)$ and DUAL$(A, c; w)$ in [9, 10], and our basic algorithm (12–13) reduces to the dual algorithm (7–8), provided we take $q_l(p_l(t)) = c_l$ in (7) though this choice of $q_l(\cdot)$ does not satisfy certain conditions required for the stability proof in [10]. The nonnegativity constraint on our price adjustment (12) complicates stability analysis considerably, especially for the asynchronous case where communication between links and sources are uncoordinated, feedback delays are substantial and time–varying, and computations can be based on outdated information [14]. Without the constraint, the same argument as in [10] using the dual objective function $D(p)$ as the Lyapunov function implies global stability for all stepsize $\gamma > 0$ in (12). With the constraint, the dual objective function $D(p)$ serves as a Lyapunov function *provided the stepsize γ is sufficiently small.* The marking scheme in [6] implements the primal algorithm (5–6). The marks per unit flow $p^s(t)$ a source receives is the shadow price at the single bottleneck link. There, shadow price is defined as the marginal increment in expected loss with a marginal increment in load. With REM, the marks allow a source s to estimate the price $p^s(t)$ which is the sum of shadow prices $p_l(t)$ at links l in the path of s. Here, however, shadow price is defined to be the marginal increase in aggregate source utility with a marginal increase in link capacity. In view of the remark above, REM can also be regarded as a marking implementation of the dual algorithm (7–8) with log utility functions.

3.4 Minimum cost flow control

The previous two approaches require binary feedback for implementation. The next two do not.

The approach taken in [7] reformulates the welfare maximization (2–3) by absorbing the capacity constraint (3) into the objective function as a cost of violating the constraint. It solves:

$$\min_{x_s \geq 0} \quad \sum_l g_l(x^l) - \sum_s U_s(x_s) \tag{14}$$

where $x^l = \sum_{s \in S(l)} x_s$ is the aggregate source rate at link l, and $g_l(x^l)$ measures the cost for link l to carry a flow of x^l. For instance g_l can be a monotonic function of link loss or delay. We assume $g_l(\cdot)$ is convex increasing: larger flow costs more. The Kuhn–Tucker condition then implies that x is the optimal rate vector if and only if the marginal utility of a source is less or equal to the marginal cost of carrying its flow through the network:

$$U_s'(x_s) \quad \leq \quad \sum_{l \in L(s)} g_l'(x^l)$$

with equality if $x_s > 0$. Hence a positive flow produces a marginal utility equal to the marginal cost. Gradient projection algorithm to solve (14) is:

$$x_s(t+1) \quad = \quad \left[x_s(t) + \gamma \left(U_s'(x_s(t)) - \sum_{l \in L(s)} g_l'(x^l(t)) \right) \right]^+ \tag{15}$$

This algorithm is referred to as the minimum cost flow control (MCFC) in [7]. It adjusts source rate to equalize marginal utility $U_s'(x_s(t))$ with the marginal cost of the path $\sum_{l \in L(s)} g_l'(x^l(t))$, and converges to an optimal rate vector provided the stepsize $\gamma > 0$ is sufficiently small.

To implement algorithm (15) without explicit feedback, choose the link cost function $g_l(x^l)$ such that its derivative $g_l'(x^l)$ is the loss probability at link l, as a function of carried flow x^l. Then, when the loss probability is small, the end–to–end loss probability $\hat{\lambda}^s = 1 - \Pi_{l \in L(s)}(1 - g_l'(x^l))$ for a packet of source s is approximately the sum $\lambda^s = \sum_{l \in L(s)} g_l'(x^l)$ of link loss probabilities in the path. Hence this quantity λ^s, which is needed for rate adjustment (15), can be estimated at the source by observing packet losses. This leads to the following stochastic gradient realization of algorithm (15):

$$x_s(t+1) \quad = \quad \begin{cases} [x_s(t) + \gamma(U_s'(x_s(t)) - 1)]^+ & \text{if packet loss} \\ x_s(t) + \gamma U_s'(x_s(t)) & \text{else} \end{cases} \tag{16}$$

where the gradient $U_s'(x_s(t)) - \lambda^s(t)$ is approximated by (the marginal utility minus) the observed loss at source s.[2] Note that loss is observed at source s with probability $\hat{\lambda}^s(t)$.

Suppose further that we choose the utility functions $U_s(x_s)$ to be such that the marginal utility is

$$U_s'(x_s) \quad = \quad \frac{\eta}{\eta + x_s^2}$$

for some $\eta > 0$. Then in equilibrium the successive change Δx_s in source rate is zero on average, and hence, roughly,

$$E\{\Delta x_s\} \quad = \quad \hat{\lambda}^s \cdot \gamma(U_s'(x_s(t)) - 1) \; + \; (1 - \hat{\lambda}^s) \cdot \gamma U_s'(x_s(t)) \quad = \quad 0$$

[2] In fact, as pointed out in [7], source rate $x_s(t)$ must be bounded away from zero in order to receive feedback from the network; we are ignoring this effect to keep the notation simple.

where $\hat{\lambda}^s$ is the (true) end–to–end loss probability. This implies that

$$\frac{\eta}{\eta + x_s^2} = \hat{\lambda}^s$$

Hence under MCFC equal rates are allocated to sources with the same end–to–end loss probability, and the equilibrium rates x_s are related to the square root of loss probability $\sqrt{\hat{\lambda}^s}$, consistent with observation on the current Internet.

Relating window to rate, $W_s(t) = D_s(t)x_s(t)$, where $W_s(t)$ is the window size and $D_s(t)$ is the round trip time, algorithm (16) translates to one that directly updates the window size:

$$W_s(t+1) = \begin{cases} \left[W_s(t) + \gamma D_s(t)(U_s'(\frac{W_s(t)}{D_s(t)}) - 1)\right]^+ & \text{if packet loss} \\ W_s(t) + \gamma D_s(t)U_s'(\frac{W_s(t)}{D_s(t)}) & \text{else} \end{cases}$$

3.5 (p, α)–proportional fairness

The paper [15] asks the fundamental question of whether source rates are well defined under window flow control, using an interesting model that explicitly incorporates round trip delays and first–in–first–out service discipline. They then generalize proportional fairness and relates it to the solution of the welfare maximization (2–3) with a specific welfare function. Finally they propose a window adjustment scheme that converges to the solution. We now elaborate.

Let q_l be the ratio of the equilibrium buffer occupancy to link capacity c_l at link l, and let $Q = diag(q_l)$ be the $L \times L$ diagonal matrix with q_l as the diagonal elements. Let $X = diag(x_s)$ be the $S \times S$ diagonal matrix with the source rates x_s as the diagonal elements. For source s, let d_s be the round–trip propagation delay (excluding queueing delay), and w_s be the window size in equilibrium. Then their model is described by the following equations on the equilibrium situation:

$$Ax \leq c \tag{17}$$

$$Q(Ax - c) = 0 \tag{18}$$

$$X(A^T q + d) = w \tag{19}$$

$$x \geq 0, \quad q \geq 0 \tag{20}$$

The inequality (17) is the capacity constraint (3). In equilibrium, either the queue length $q_l = 0$ or the aggregate rate at link l match the capacity, i.e., $x^l = \sum_{s \in S(l)} x_s = c_l$. This is expressed as identity (18), $q_l(x^l - c_l) = 0$ for all l. Identity (19), which equates the window size with the total amount of fluid in transit, is the most interesting. With first–in–first–out, the fraction of fluid buffered at link l that belongs to source s is proportional to x_s/c_l *in equilibrium.* Be definition of q_l, this amount is $q_l x_s$, and hence $x_s \sum_{l \in L(s)} q_l$ is the amount buffered at all links in the path of s. This plus the amount $x_s d_s$ of fluid in propagation in between links must equal the window size in equilibrium. This is expressed by the identity (19), $x_s(\sum_{l \in L(s)} q_l + d_s) = w_s$ for all s. The

first main result of [15], proved using fixed point theorem and Farkas Lemma, is that, given any (w, A, d, c), there exists a unique rate vector, denoted $x(w)$, that satisfies (17–20). Indeed this can be proved by considering our welfare maximization (2–3) with the following concave increasing (since $x_s \leq w_s/d_s$ in reality) utility functions:

$$U_s(x_s) \quad = \quad w_s \log x_s - d_s x_s \tag{21}$$

Then (17–20) can be verified as the Kuhn–Tucker condition for x to be a solution of the welfare maximization (2–3). As noted in Section 3.1, such a solution exists and is unique since the welfare function is strictly concave and the feasible solution set is compact.

Let $p = (p_s, s \in S)$ be a positive vector and α a positive number. A vector x^* is said to be (p, α)–proportionally fair if it is feasible and for any other feasible vector x,

$$\sum_s p_s \frac{x_s - x_s^*}{x_s^{*\alpha}} \quad \leq \quad 0$$

This reduces to (4) when $\alpha = 1$. It can be verified using Kuhn–Tucker condition that x^* is (p, α)–proportionally fair if and only if it solves the welfare maximization (2–3) with utility functions, for $\alpha > 0$,

$$U_s(x_s) \quad = \quad \begin{cases} p_s \log x_s & \text{if } \alpha = 1 \\ p_s (1-\alpha)^{-1} x^{1-\alpha} & \text{else} \end{cases} \tag{22}$$

The vector p in the fairness definition is related to the sources' backlogs along their paths in equilibrium. Specifically, given a target backlog $(p_s, s \in S)$, it is shown in [15] that there is a unique window vector $w = (w_s, s \in S)$ under which the total backlog $w_s - x_s d_s$ of source s equals p_s in equilibrium. Moreover the resulting equilibrium rate $x(w)$ is $(p, 1)$–proportionally fair.

Recall from Section 2.2 that TCP vegas adjusts its window size so as to stabilize the total backlog in the path to be around a target value p. This suggests that TCP vegas attempts to achieve $(p, 1)$–proportionally fair allocations. In order words, it solves welfare optimization (2–3) with utility functions (22) with $\alpha = 1$.

To achieve $(p, 1)$–proportional fairness, [15] proposes the following window adjustment scheme

$$\frac{d}{dt} w_s(t) \quad = \quad -\gamma \frac{d_s}{D_s(t)} \left(\frac{p_s}{w_s(t)} + \frac{d_s}{D_s(t)} - 1 \right) \tag{23}$$

where d_s is the round trip propagation delay, and $D_s(t)$ is the round trip time including queueing delay. Hence the algorithm only requires measurement of round trip time $D_s(t)$, provided source s knows or estimates, as TCP vegas does, its round trip propagation delay. Using a Lyapunov argument, algorithm (23) can be shown to converge to the unique window vector that yields $(p, 1)$–proportionally fair rates.

4 Conclusion

We have briefly reviewed four flow control proposals for the Internet based on welfare optimization. These algorithms are globally stable under mild conditions on the utility functions, and are simple to implement, requiring only binary or implicit feedback. As noted in the Introduction, by treating practical flow control schemes as distributed implementations of an optimization algorithm, the framework here allows a systematic refinement where modifications to a flow control scheme are guided by enhancements to the optimization algorithm. This would be particularly valuable for Internet flow control which must endure severe feedback delay and asynchronism.

In contrast with TCP, where link algorithm (RED) evolves independently of source algorithms (TCP tahoe, reno, and vegas), the link and source algorithms with optimization flow control must be designed jointly and operate in concert with each other. This can be arranged in a cooperative environment, such as virtual private networks. In a noncooperative environment, such as the public Internet, congestion pricing provides the right incentive and the necessary information for sources to react in a socially optimal manner.

References

[1] Sanjeewa Athuraliya and Steven Low. Optimization flow control with Newton–like algorithm. In *Proceedings of IEEE Globecom'99*, December 1999.

[2] 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.

[3] Richard Edell and Pravin Varaiya. Prividing Internet access: what we learn from INDEX. Preprint, 1999.

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

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

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

[7] 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.

[8] V. Jacobson. Congestion avoidance and control. *Proceedings of SIGCOMM'88, ACM*, August 1988. An updated version is available via ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

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

[10] 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.

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

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

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

[14] Steven H. Low and David E. Lapsley. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 1999. To appear.

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

[16] 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.