A Mathematical Framework for Designing a Low-Loss, Low-Delay Internet

Steven H. Low
Computer Science and Electrical Engineering
California Institute of Technology
slow@caltech.edu

R. Srikant
Coordinated Science Lab. and General Engineering
University of Illinois at Urbana-Champaign
rsrikant@uiuc.edu

November 29, 2002

Abstract

We survey some recent results on modeling, analysis and design of congestion control schemes for the Internet. Using tools from convex optimization and control theory, we show that congestion controllers can be viewed as distributed algorithms for achieving fair resource allocation among competing sources. We illustrate the use of simple mathematical models to analyze the behavior of currently deployed Internet congestion control protocols as well as to design new protocols for networks with large capacities, delays and general topology. These new protocols are designed to nearly eliminate loss and queueing delay in the Internet, yet achieving high utilization and any desired fairness.

1 Introduction

Over the last decade and a half, the Internet has grown from a small collection of nodes connecting primarily academic institutions to a pervasive network which is at the center of world-wide commerce. A significant technological breakthrough which facilitated this growth was the introduction of congestion control (Jacobson, 1988) that allowed many users to share the network without causing congestion collapse. However, the Internet as it is today does not guarantee a high quality-of-service to its users. One of the reasons is that users in the Internet estimate the level of congestion by measuring packet loss or delay. Thus, a *bad* event (either high loss or high delay) has to occur before users can infer network congestion. To counter this problem, in the last few years, there has been a surge of interest in the Internet community to provide low-loss, low-delay service by encouraging users to adapt to network congestion based on feedback from the routers in the form of Explicit Congestion Notification (ECN) (Floyd, 1994). ECN marks allow routers to notify users about incipient congestion. A packet is said to be marked if a particular bit in its header is set to one. The user who receives a marked packet reacts to the mark by cutting its transmission rate. Thus, the router avoids dropping packets, thereby enhancing goodput (throughput minus retransmissions), and still manages to convey congestion information to the user. This has the potential to provide real-time services with minimal additional burden on the core network i.e., without the need of a centralized admission control, resource reservation or complicated scheduling mechanisms (Key, Kelly and Zachary, 2000).

To provide ECN marks in a way that ensures fairness while maintaining small queue lengths, the routers have to intelligently select the packets to mark in a manner that conveys information about the current state of the network to the users. Algorithms which the routers employ to convey such information are called Active Queue Management (AQM) schemes. The parameters of the AQM scheme have to be designed to ensure low-loss network operation. Of course, it is easy to ensure that there is no packet loss by marking every packet, which would, in turn, force the sources not to transmit any packets at all! Thus, in addition to requiring a low-loss network operation, we also require that the network utilization be high. In general, the choice of the AQM parameters to tradeoff between these requirements (low-loss and high utilization) may depend upon the current state of the network: number of users in the network, their routes, and link capacities, etc. However such information is not available at the routers in the network. Thus, the focus of this paper is to review some recent literature on the use of simple mathematical models to design scalable congestion control schemes at the sources and AQM schemes at the routers to eliminate loss and queueing delay in the Internet. The interested reader is also referred to (Srikant, 2000) for a general discussion on control issues in communication networks and alternate schemes for congestion control in ATM networks, and to (Low, Paganini and Doyle, 2002) for an earlier review of Internet congestion control.

The rest of the paper is organized as follows. In Section 2, we show the connection between resource allocation and congestion control and review results that show that congestion control algorithms can be viewed as decentralized schemes to solve a *fair* resource allocation problem. In Section 3, we discuss some widely used versions of TCP and AQM schemes in the Internet and interpret them in terms of the models introduced in Section 2. In Sections 2–3, our emphasis is on fair resource allocation and distributed algorithms to achieve this equilibrium. Thus, we interpret congestion control using the language of convex optimization. To understand the effect of feedback delay on the stability of this equilibrium requires control-theoretic analysis. In Section 4, we extend the network model to incorporate delay, review the stability of the current protocols, and briefly describe some ideas behind the design of scalable congestion control protocols that can maintain linear stability, achieve high utilization, while virtually eliminating loss and queueing delay in the network. The models used to design these protocols are linear, deterministic abstractions of the true behavior of the Internet. We address these modeling assumptions in Section 5. In Section 6, we extend the class of congestion control algorithms we considered to include multicast (point-to-multipoint) communications. Concluding remarks are provided in Section 7.

2 Resource Allocation and Congestion Control

Consider a large network shared by many users, where the goal is to share the network resources in a *fair* manner. The network resources that we consider here are the link bandwidths. There is no universally accepted definition of fairness; in this paper, we will associate a utility function with each user in the network and we will refer to a resource allocation scheme as being fair if it maximizes the sum of utilities of all the users in the network. For an alternate definition of fairness, see (Goel, Meyerson and Bharghava, 2001).

We will use the terms"user", "source", and "connection" interchangeably.

2.1 Problem formulation

A network is modeled as a set of resources indexed by l, called links, with finite capacities c_l. It is shared by a set of sources, indexed by i. Let $U_i(x_i)$ be the utility of source i as a function of its rate x_i (measured in packets per unit time). Associated with each source is a route which is a collection of links in the network. Let R be a *routing matrix* whose (l, i) entry is 1 if source i's route includes link l and is 0 otherwise.

The resource allocation problem can be formulated as the following nonlinear program (Kelly, 1997):

$$\max_{x \geq 0} \ \sum_i U_i(x_i), \qquad Rx \leq c \tag{1}$$

where x is the vector of source rates and c is the vector of link capacities. The constraint says that, at each link l, the aggregate source rate $\sum_i R_{li} x_i$ does not exceed the capacity c_l. If the utility functions are strictly concave, then the above nonlinear program has a unique optimal solution.

To solve this problem directly, we have to the know the utility functions and routes of all the sources in the network. In a large network such as the Internet, this information is not available centrally. Thus, it is important to devise distributed solutions, where each source adapts its transmission rate based only on local information. In the rest of this section, we will describe distributed algorithms whereby the sources adapt their rates based on congestion feedback from the links in their paths.

2.2 Penalty function approach

Instead of trying to obtain the optimal solution to problem (1), consider the following problem where the constraints are added to the objective using penalty functions:

$$\max_{x \geq 0} \ \sum_i U_i(x_i) - \sum_l \int_0^{\sum_{i:l \in i} x_i} p_l(x) dx \tag{2}$$

where we use the notation $l \in i$ to indicate that link l belongs to source i's route. The function $p_l(x)$ is assumed to be non-negative and denotes the penalty function corresponding to the capacity constraint at link l. This is commonly referred to as *price* at link l. Note that the price at link l is a function of the total arrival rate $\sum_{i:l \in i} x_i$ at link l, and can be interpreted as a measure of congestion at link l.

To solve (2), we write down the first-order necessary condition: for each i,

$$U_i'(x_i) - \sum_{l \in i} p_l(\sum_{j:l \in j} x_j) = 0$$

Thus, the following continuous-time version of a *gradient-ascent* algorithm can be used to solve (2):

$$\dot{x}_i = \kappa_i \left(1 - \frac{1}{U_i'(x_i)} \sum_{l \in i} p_l \left(\sum_{j : l \in j} x_j \right) \right) \tag{3}$$

Note that the equilibrium point of the congestion control equation (3) solves (2). To prove that the system of equations given by (3) is stable, it was shown in (Kelly, Maulloo and Tan, 1998) that the objective in (2) is a Lyapunov function for the system of equations given by (3). Specifically, if we denote the objective in (2) by $V(x)$, then it is easy to verify that

$$\frac{dV}{dt} = \sum_i \frac{\partial V}{\partial x_i} \dot{x}_i$$

which is strictly positive when x is not an equilibrium point of (3), and zero otherwise. The algorithm (3), with appropriate price function $p_l(\cdot)$ (see below), is referred to as the *primal algorithm.*

The most important feature of (3) is that source i's congestion control equation depends only on the sum of the link prices along its path. Thus, from each source's point of view, if there is a protocol that allows it to compute the sum of the link prices along its path, then it can implement its congestion controller in a distributed manner, without requiring any coordination with other sources in the network. From the network's point of view, the functions $p_l(\cdot)$ will be computed by the routers in the network. The fact that the price of link l depends only on the total arrival rate at the link ensures that the computations at the routers depend on the aggregate flow into each link. In other words, no *per-flow* information is necessary. Further, each link's computation can be performed without requiring any coordination with other links. Thus, the algorithm is completely decentralized, except for the requirement of a protocol to communicate the link prices to the sources. We will later describe a simple protocol for this purpose.

2.3 Dual approach

Instead of solving the optimization problem (1) using penalty functions, we can also solve the problem exactly using Lagrange multipliers. Consider the associated dual problem (Low and Lapsley, 1999):

$$\min_{\lambda_l \geq 0} \qquad D(\lambda) := \sum_i \max_{x_i \geq 0} \left(U_i(x_i) - x_i \sum_{l \in i} \lambda_l \right) \; + \; \sum_l \lambda_l c_l \tag{4}$$

Given the Lagrange multipliers λ_l (also referred to as *shadow prices*), the maximization over x_i can be carried out by individual sources based only on the aggregate price of its path, as follows:

$$x_i \quad = \quad U_i'^{-1}\left(\sum_{l\in i}\lambda_l\right) \tag{5}$$

where $U_i'^{-1}(\cdot)$ denotes the inverse of the derivative of U_i. When the utility function U_i is convex, x_i is a decreasing function of the aggregate price $\sum\lambda_l$.

To compute the shadow prices, consider the continuous-time version of the scaled gradient projection algorithm for the dual problem (4):

$$\dot{\lambda}_l \quad = \quad -\gamma_l\frac{\partial D}{\partial\lambda_l}$$

where γ_l is a stepsize parameter at link l. It is easy to verify that

$$\frac{\partial D}{\partial\lambda_l} = c_l - \sum_{i:l\in i} x_i$$

Thus, an algorithm to compute the shadow price at link l becomes

$$\dot{\lambda}_l = \gamma_l\left(\sum_{i:l\in i} x_i - c_l\right) \tag{6}$$

This is simply the law of supply and demand: if the demand $\sum x_i$ exceeds supply c_l, increase the price; otherwise, decrease it. Again, we note the distributed nature of the algorithm: each link only uses the total arrival rate into it to compute its price. The global convergence of the above algorithm has been proved in (Low and Lapsley, 1999), even in an asynchronous setting where delays between sources and links can be large and time-varying, sources and links can communicate and update at different times, with different frequencies, and using outdated information. The algorithm (6) and (5) is referred to as the *dual algorithm*.

When c_l in (6) is the real link capacity, the equilibrium queue length at link l is p_l^*/γ_l where p_l^* is the equilibrium price at link l. Not only can it be large (when γ_l is small), worse still, p_l^* and hence the equilibrium queue length typically increases as the number of sources increases. One way to maintain zero equilibrium queue regardless of the number of sources is to take c_l to be strictly less than real link capacity. Another way is to explicitly incorporate queue length control into (6) (Athuraliya, Li, Low and Yin, 2001). The global stability of such an explicit queue length control in the absence of delay has been established in (Paganini, 2002) using a Lyapunov argument. It is well known that Newton algorithm generally has much faster convergence than gradient projection algorithm. In (Athuraliya and Low, 2000), dynamic scaling factors $\gamma_l(t)$ are used to approximate the Newton algorithm. This approximate Newton algorithm maintains optimality, has the same decentralization structure as (6), but enjoys a better convergence property.

2.4 Exact penalty functions

The penalty function approach provides an approximate solution to the resource allocation problem (1). By comparing the first-order necessary conditions for optimality, it is easy to see that the solution to (2) will solve (1) *exactly* if

$$p_l\left(\sum_{j:l\in j} x_j\right) = \lambda_l$$

at the optimal solution of (2). Recall the penalty $p_l(\cdot)$ is computed at each link based on the total arrival rate $\sum_{i:l\in i} x_i$ into the link. An example of such a penalty function is

$$p_l(x_l) = \left(\frac{x_l}{\tilde{c}_l}\right)^B$$

for some $\tilde{c}_l$. Such a penalty function can be interpreted as the probability that the queue exceeds a level B in an $M/M/1$ queue with arrival rate x_l and service rate $\tilde{c}_l$ when $x_l < \tilde{c}_l$ (Gibbens and Kelly, 1999). We will refer to $\tilde{c}_l$ as the *virtual capacity* of the link. Another example of a penalty function is

$$p_l(x_l) = \frac{(x_l - \tilde{c}_l)^+}{x_l}$$

This penalty function can be interpreted as the loss probability in an $M/M/1/B$ queue when $B \to 0$ (Kunniyur and Srikant, 2000). Since the penalty function approach only approximately solves the resource allocation problem (1), a question of both theoretical and practical interest is whether one can choose $\{\tilde{c}_l\}$ such that the solution to (2) solves (1) exactly. Towards this end, consider the following differential equations for $\tilde{c}_l$:

$$\dot{\tilde{c}}_l = \alpha(c_l - \sum_{j:l\in j} x_j), \tag{7}$$

with the understanding that $\tilde{c}_l \geq 0$ and α is a small parameter that can be chosen to stabilize the system. Noting that the penalty functions are decreasing functions of $\tilde{c}_l$, the above differential equation can be interpreted as follows: *if the arrival rate at a link is less than the link capacity, then decrease the penalty by increasing $\tilde{c}_l$; else, increase the penalty by decreasing $\tilde{c}_l$.* A modified version of this algorithm, along with the primal algorithm (3), has been shown to converge to the solution of the network resource allocation problem exactly (Kunniyur and Srikant, 2002c) when α is sufficiently small. Again, we wish to emphasize the decentralized nature of the algorithm: each link computes its virtual capacity knowing only the total arrival rate into it.

2.5 One-bit feedback

So far, we have assumed that each link on a source's path computes a penalty or price for using that path and the sum of the prices of the links in its path is conveyed to each source. One way to implement this would be as follows (Lapsley and Low, 1998): suppose that each packet has a field for storing price information. This could be initialized to zero at the source. Then, as the packet traverses the network, each link can add its price to this field. The destination can then return the total path price to the source upon receiving each packet. However, in today's Internet, such a price field is not available in the IP packet header. In fact, the proposed standards for conveying congestion information, use only one bit to indicate congestion (Ramakrishnan and Floyd, 1999)! The congestion-indication bit, known as the ECN bit, is initially set to zero at the source. When a router wants to indicate congestion, it sets the bit to one. This is referred to as *packet marking.* We now describe two one-bit marking schemes to convey path prices, REM (Random Exponential Marking) (Athuraliya et al., 2001) and SAM (Self-normalized Additive Marking) (Adler, Cai, Shapiro and Towsley, 2002)). They can be used for both the primal and the dual algorithm.

Let $p_l \geq 0$ be the link prices. When a packet arrives at link l, if it is already marked, the ECN bit is forwarded as is. Otherwise, it is marked with probability $1 - e^{-p_l}$. Then the end-to-end probability that a packet will be marked on a path is $1 - exp(-\sum p_l)$, where the summation is over all the links in the path. Thus, the aggregate price $\sum p_l$ is embedded in the (exponent of the) end-to-end marking probability. If the destination conveys the rate at which packets are marked to the source, then the source can compute the end-to-end path price by $-\ln(1-m)$, where m is the end-to-end marking probability. This is REM (Athuraliya et al., 2001).

Suppose now the prices are bounded, say, $0 \leq p_l \leq 1$. Suppose further that every link knows its position along a packet's path, and without loss of generality, assume link l, $l = 1, \ldots, n$ is the l^{th} link in the path under consideration. Then when a packet arrives at link l, its ECN bit is forwarded as is to link $l+1$ with probability $1 - 1/l$, is set to 1 with probability p_l/l, and is set to 0 with probability $(1-p_l)/l$. Then the end-to-end marking probability is $\sum_{l=1}^{n} p_l/n$. In IP networks, link position can usually be inferred from TTL (time-to-live) filed in the IP header of a packet. This is

SAM (Adler et al., 2002).

Even though estimation of the path price both converges almost surely to its true value, REM is a biased estimator (since the ln function is strictly concave) whereas SAM is unbiased. It is shown in (Adler et al., 2002) that if there are no other restriction on prices except $p_l \geq 0$, then REM is essentially the only scheme that can convey prices with a single bit. If prices are bounded and links know their positions in a packet's path, then REM and SAM are essentially the only possible schemes. In this case, SAM generally outperforms REM.

3 Current TCP/AQM Protocols

In the previous section, we have shown that congestion control is a distributed algorithm to share network resources among competing sources. It has two components: a source algorithm that dynamically adjusts rate (or window size) in response to congestion in its path, and a link algorithm that updates, implicitly or explicitly, a congestion measure and sends it back, implicitly or explicitly, to sources that use that link. We have also presented the primal and the dual algorithms, and their variations, as examples. These algorithms are derived from the source utility functions.

In the current Internet, the source algorithm is carried out by TCP, and the link algorithm is carried out by (active) queue management (AQM) schemes such as DropTail or RED (Floyd and Jacobson, 1993). Different protocols use different metrics to measure congestion, e.g., TCP Reno (Jacobson, 1988), and its variants, use loss probability as a congestion measure, and TCP Vegas (Brakmo and Peterson, 1995) uses queueing delay as a congestion measure (Low, Peterson and Wang, 2002). Both are implicitly updated at the links and implicitly fed back to sources through end-to-end loss or delay, respectively. The equilibrium and dynamics of the network depends on the TCP/AQM protocol pair.

In this section, we will briefly review major TCP (Section 3.1) and AQM protocols (Section 3.2), and present mathematical models of these protocols (Section 3.3). Even though these protocols were not designed to solve any global optimization problem, we show how to associate utility functions with any TCP protocol (Section 3.4), thus allowing us to interpret them as different distributed algorithms to solve the utility maximization problem formulated in Section 2; see (Low, 2000) for more details.

3.1 TCP algorithms

TCP uses "window" flow control, where a destination sends acknowledgments for packets that are correctly received. A source keeps a variable called *window size* that determines the maximum number of outstanding packets that have been transmitted but not yet acknowledged. When the window size is exhausted, the source must wait for an acknowledgment before sending a new packet. Two features are important. The first is the "self-clocking" feature that automatically slows down the source when a network becomes congested and acknowledgments are delayed. The second is that the window size controls the source rate: roughly one window of packets is sent every round-trip time. The first feature was the only congestion control mechanism in the Internet before Jacobson's proposal in 1988 (Jacobson, 1988). Jacobson's idea is to *dynamically* adapt window size to network congestion. In this section, we will review how TCP infers congestion and adjusts window size.

3.1.1 TCP Tahoe and Reno

A connection starts cautiously with a small window size of one packet (up to four packets have recently been proposed) and the source increments its window by one every time it receives an acknowledgment. This doubles the window every round-trip time and is called `slow-start`. When the window reaches a threshold, 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 acknowledgment. This increases the window by one in each round-trip time, and is referred to as additive increase. The threshold that determines the transition from `slow-start` to `congestion avoidance` is meant to indicate the available capacity in the network and is adjusted each time a loss is detected. On detecting a loss, the source sets the slow-start threshold to half of the current window size, retransmits the lost packet, and re-enters `slow-start` by resetting its window to one.

This algorithm was proposed in (Jacobson, 1988) and implemented in the Tahoe version of TCP. Two refinements, called `fast recovery`, were subsequently implemented in TCP Reno to recover from loss more efficiently. Call the time from detecting a loss (through duplicate acknowledgments) to receiving the acknowledgment for the retransmitted packet the `fast retransmit/fast recover (fr/fr)` phase. In TCP Tahoe, the window size is frozen in the `fr/fr` phase. This means that a new packet can be transmitted only a round-trip time later. Moreover, the "pipe" from the source to the destination 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 first refinement allows a Reno source to temporarily increment its window by one on receiving each duplicate acknowledgment while it is in the `fr/fr` phase. The rationale is that each duplicate acknowledgment signals that a packet has left the network. When the window becomes larger than the number of outstanding packets, a *new* packet can be transmitted in the `fr/fr` phase while it is waiting for a (nonduplicate) acknowledgment for the retransmitted packet. The second refinement essentially sets the window size at the end of the `fr/fr` phase to half of the window size when `fr/fr` starts and enters `congestion avoidance` directly. Hence, `slow-start` is entered only rarely in TCP Reno when the connection first starts and when a loss is detected by timeout rather than duplicate acknowledgments.

3.1.2 TCP Vegas

TCP Vegas (Brakmo and Peterson, 1995) improves upon TCP Reno through three main techniques. The first is a modified retransmission mechanism where timeout is checked on receiving the first duplicate acknowledgment, rather than waiting for the third duplicate acknowledgment (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 it results in fewer losses.

The third technique is a new congestion avoidance mechanism that corrects the oscillatory behavior of Reno. The 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 to whether 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. Another interpretation of the congestion avoidance algorithm of Vegas is given in (Low, Peterson and Wang, 2002), in which a Vegas source periodically measures the round-trip *queueing* delay and sets its rate to be proportional to the ratio of its round trip propagation delay to queueing delay, the proportionality constant being between α and β. Hence, the more congested its path is, the higher the queueing delay and the lower the rate. The Vegas source obtains queueing delay by monitoring its round-trip time (the time between sending a packet and receiving its acknowledgment) and subtracting from it the round-trip propagation delay.

3.2 AQM algorithms

3.2.1 DropTail

Congestion control of the Internet was entirely source-based at the beginning, in that the link algorithm was implicit. A link (router) simply drops a packet that arrives at a full buffer. This is called DropTail (or Tail Drop) and the implicit link algorithm is carried out by the queue process. The congestion measure it updates depends on the TCP algorithm.

For TCP Reno and its variants, the congestion measure is packet loss probability. The end-to-end loss probability is observed at the source and is a measure of congestion on the end-to-end path. For TCP Vegas, the congestion measure turns out to be link queueing delay (Low, Peterson and Wang, 2002) when first-in-first-out service discipline is used. The congestion measure of a path is the sum of queueing delays at all constituent links.

3.2.2 RED

RED (Random Early Detection) (Floyd and Jacobson, 1993) is an alternative way to generate packet loss as a congestion measure. Instead of dropping only at a full buffer, RED maintains an exponentially weighted queue length and drops (or marks) packets with a probability that increases with the average queue length. When the average queue length is less than a minimum threshold, no packets are dropped. When it exceeds a maximum threshold, all packets are dropped. When it is in between, a packet is dropped with a probability that is a piecewise linear and increasing function of the average queue length.

We also note that other AQM schemes have been proposed recently. Some notable examples include the PI controller (Hollot, Misra, Towsley and Gong, 2001b), REM (Athuraliya et al., 2001) and AVQ (Kunniyur and Srikant, 2001).

3.3 Mathematical models of current protocols

As an illustration of the framework presented in Section 2, we now present models of Reno/RED and Vegas/DropTail. We will discuss the implications of these models in the following subsection.

3.3.1 Reno/RED

We only model the average behavior of AIMD (the additive increase, multiplicative decrease algorithm used to control the window size) and does not differentiate between TCP Reno (Stevens, 1999) and its variants such as NewReno, SACK, etc. All these protocols (henceforth referred to as 'Reno') increase the window by one every round trip time if there is no loss in the round trip time, and halves the window otherwise. There are two versions of multiplicative decrease. Let $w_i(t)$ be the window size. Let τ_i be the equilibrium round trip time (propagation plus equilibrium queueing delay), which we assume is constant. Let $x_i(t)$ defined by $x_i(t) = w_i(t)/\tau_i$ be the source rate at time t. The time unit is on the order of several round trip times and source rate $x_s(t)$ should be interpreted as the average rate over this timescale. Dynamics smaller than the timescale of a round trip time is not captured by the fluid model.

Let $p_l(t)$ be the marking probability at link l at time t. We make the key assumption that the end-to-end marking probability $q_i(t)$ to which source algorithm reacts is the sum of link marking probabilities: $q_i(t) = \sum_l R_{li} p_l(t)$.[1] This is reasonable when $p_l(t)$ are small, in which case $q_i(t) = 1 - \prod_l (1 - p_l(t))^{R_{li}} \simeq \sum_l R_{li} p_l(t)$. In period t, it transmits at rate $x_i(t)$ packets per unit time, and receives (positive and negative) acknowledgments at approximately the same rate, assuming every packets is acknowledged. Hence, on the average, source i receives $x_i(t)(1-q_i(t))$ number of positive acknowledgments per unit time and each positive acknowledgment increases the window $w_i(t)$ by $1/w_i(t)$. It receives, on the average, $x_i(t)q_i(t)$ negative acknowledgments (losses) per unit time and each halves the window. Hence, in period t, the net change to the window is roughly

$$x_i(t)(1 - q_i(t))/w_i(t) - x_i(t)q_i(t)w_i(t)/2$$

Then Reno is modeled by the source algorithm F_i:

$$\dot{x}_i = \frac{1 - q_i(t)}{\tau_i^2} - \frac{1}{2} q_i(t) x_i^2(t) \tag{8}$$

The quadratic term signifies the property that, if rate doubles, the multiplicative decrease occurs at twice the frequency with twice the amplitude. This model is used in (Kelly, 2001) and (Low, 2000).

There are variants of this model. One is that the window increases deterministically by 1 every round trip time. This modifies the additive increase term in (8) into:

$$\dot{x}_i = \frac{1}{\tau_i^2} - \frac{1}{2} q_i(t) x_i^2(t) \tag{9}$$

This is used in (Kunniyur and Srikant, 2000; Hollot et al., 2001b; Massoulie and Roberts, 1999). It is approximately the same as (8) when loss probabilities $p_l(t)$ are small.

[1] If this assumption is violated, the model presented below still holds, but the underlying utility function is modified; see (Kelly, 2001, Theorem 4).

Another variant is that, instead of halving the window on each negative acknowledgment, the window is halved once in each round trip time that contains one or more negative acknowledgments. This modifies the multiplicative decrease term in (8) into:

$$\dot{x}_i \;=\; \frac{1-q_i(t)}{\tau_i^2} - \frac{1}{2\tau_i} q_i(t) x_i(t) \tag{10}$$

This is studied in (Low, 2000).

RED (Floyd and Jacobson, 1993) maintains two internal variables, the instantaneous queue length $b_l(t)$ and average queue length $r_l(t)$. They are updated according to

$$\dot{b}_l \;=\; y_l(t) - c_l \tag{11}$$

$$\dot{r}_l \;=\; -\alpha_l (r_l(t) - b_l(t)) \tag{12}$$

where $\alpha_l \in (0,1)$. Then, (the 'gentle' version of) RED marks a packet with a probability $p_l(t)$ that is a piecewise linear increasing function of $r_l(t)$:

$$p_l(t) \;=\; \begin{cases} 0 & r_l(t) \le \underline{b}_l \\ \rho_1(r_l(t) - \underline{b}_l) & \underline{b}_l \le r_l(t) \le \overline{b}_l \\ \rho_2(r_l(t) - \overline{b}_l) + m_l & \overline{b}_l \le r_l(t) \le 2\overline{b}_l \\ 1 & r_l(t) \ge 2\overline{b}_l \end{cases} \tag{13}$$

where

$$\rho_1 \;=\; \frac{m_l}{\overline{b}_l - \underline{b}_l} \qquad \text{and} \qquad \rho_2 \;=\; \frac{1-m_l}{\overline{b}_l}$$

3.3.2 Vegas/FIFO

A model of Vegas is developed in (Low, Peterson and Wang, 2002) where queueing delay, $p_l(t) = b_l(t)/c_l$, is used as a congestion measure (see also (Mo and Walrand, 2000)). Here $b_l(t)$ is the queue length and evolves according to (11). Hence the AQM algorithm does not involve any internal variable and is given by (dividing both sides of (11) by c_l):

$$\dot{p}_l \;=\; \frac{1}{c_l}\left(y_l(t) - c_l\right) \tag{14}$$

To model the TCP algorithm, let d_i be the round trip propagation delay for source i and assume $\alpha_i = \beta_i$ for all i. Then the rate is adjusted according to:

$$\dot{x}_i \;=\; \frac{1}{(d_i + q_i(t))^2} \operatorname{sgn}\left(1 - \frac{x_i(t) q_i(t)}{\alpha_i d_i}\right) \tag{15}$$

where $\mathrm{sgn}(z)$ is -1 if $z < 0$, 0 if $z = 0$, and 1 if $z > 0$. Here, $q_i(t) = \sum_l R_{li} p_l(t)$ is the sum of link queueing delays in the path of i at time t, $d_i + q_i(t)$ is the round trip time of i at time t, and $x_i(t)q_i(t)$ is the number of packets[2] that are buffered in the queues in i's path. Hence (15) says that the window (rate × round trip time) is incremented or decremented by 1 packet per round trip time, according as the number $x_i(t)q_i(t)$ of packets buffered in the path is smaller or greater than the target $\alpha_i d_i$. In equilibrium, each source i maintains $\alpha_i d_i$ packets in its path.

Instead of the Vegas source algorithm (15), (Mo and Walrand, 2000) proposes a window adjustment scheme that achieves the same target queue and proportional fairness (Kelly et al., 1998):

$$\dot{w}_i = -\kappa \frac{d_i}{d_i + q_i(t)} \left(1 - \frac{d_i}{d_i + q_i(t)} - \frac{\alpha_i d_i}{w_i(t)} \right)$$

Note that when queueing delay $q_i(t) = 0$, $\dot{w}_i = \kappa \alpha_i d_i / w_i(t)$, i.e., when the network is not congested, window increases at a rate that is inversely proportional to current window. Global stability of the algorithm is shown in (Mo and Walrand, 2000) in the absence of feedback delay using a Lyapunov argument.

3.4 Equilibrium properties of current protocols

In this subsection, we first describe a general model of TCP/AQM algorithms, show how to associate utility functions to these algorithms, and then apply the result to Reno and Vegas models derived in the last subsection.

Define, for each link l, the aggregate flow rate $y_l(t)$ by:

$$y_l(t) = \sum_i R_{li} x_i(t) \tag{16}$$

and for each source i, the aggregate price $q_i(t)$

$$q_i(t) = \sum_i R_{li} p_l(t) \tag{17}$$

Or, in matrix form, we have $y = Rx$ and $q = R^T p$ (^{T} denotes transpose).

Then TCP algorithms can generally be expressed as:

$$\dot{z}_i = F_i(z_i, q_i) \tag{18}$$

$$x_i = G_i(z_i, q_i) \tag{19}$$

[2] This assumes $x_i(t)$ is in packets per unit time.

where z_i is a local state variable. TCP Reno and Vegas are special cases where $z_i = x_i$. AQM algorithms can generally be expressed as:

$$\dot{v}_l = H_l(y_l, v_l) \tag{20}$$

$$p_l = K_l(y_l, v_l) \tag{21}$$

For RED, v_l represents the instantaneous and average queue length. Note that decentralization requires that source rate $x_i(t)$ be adjusted based only on aggregate price $q_i(t)$, not on $p(t)$ nor $q_j(t), j \neq i$, and that price $p_l(t)$ be adjusted based only on aggregate rate $y_l(t)$, not on $x(t)$ nor $y_m(t), m \neq l$.

The equilibrium (x^*, z^*, v^*, p^*) of (18)–(21) satisfies

$$F_i(z_i^*, q_i^*) \;\; = \;\; 0$$

Assuming $\partial F_i / \partial z_i \neq 0$ in the domain of interest, then by the implicit function theorem, $z_i = f_i(q_i)$ for some f_i. Hence, eliminating z_i from (19), we get

$$x_i \;\; = \;\; G_i(f_i(q_i), q_i) \;\; =: \;\; g_i(q_i)$$

for some function g_i. We make the natural assumption that both g_i and g_i^{-1} are positive and strictly decreasing for all i, meaning that a larger congestion measure yields a smaller rate in equilibrium. Then, define the utility function of source i as

$$U_i(x_i) \;\; := \;\; \int g_i^{-1}(x_i) dx_i \tag{22}$$

that is unique up to a constant. Under the assumption on g_i^{-1}, U_i are strictly concave increasing functions. An increasing utility function implies a greedy source – a larger rate yields a higher utility – and concavity implies diminishing return.

Recall the problem of maximizing aggregate utility (1) and its dual (4). A unique optimal rate vector x^* exists since the objective function in (1) is strictly concave and the feasible solution set is compact. We can interpret TCP/AQM algorithms as distributed primal-dual procedures to solve the utility maximization problem and its dual. Specifically, suppose (x^*, p^*) is an equilibrium of (18)–(21). Then it is proved in (Low, 2000) that x^* solves the primal problem (1) with utility function given by (22) if and only if for all l

$$y_l^* \;\; \leq \;\; c_l \;\; \text{with equality if } p_l^* > 0 \tag{23}$$

Moreover, in this case, x^* is the unique primal optimal solution and p^* is a dual optimal solution, i.e., solves (4).

Hence, various TCP/AQM protocols can be modeled as different distributed primal-dual algorithms (F, G, H, K) to solve the global optimization problem (1), with different utility functions U_i. This computation is carried out by sources and links over the Internet in real time in the form of congestion control. Note that the definition of utility function U_i depends only on TCP algorithm (F_i, G_i); however, that the equilibrium (x^*, p^*) is optimal depends on AQM algorithm (H, K) through the complementary slackness condition (23), which requires that (H, K) match input rate to capacity at every bottleneck link. When this is satisfied, the gradient of the dual objective function in (4) is zero at bottleneck links, and since the dual problem is convex, p^* is necessarily a Lagrange multiplier that solves the dual problem. Any AQM that stabilizes queues possesses this property.

Applying this result to Reno algorithms (8)–(10) and to Vegas algorithm (15), the utility functions of these algorithms are derived in (Low, 2000), shown in Table 1. These utility functions are indeed strictly concave increasing and hence the equilibrium rate x^* is the unique optimal rate vector for (1). The utility function of Vegas implies that it achieves proportional fairness in equilibrium (Kelly et al., 1998).

Reno (8)	$F_i(x_i(t), q_i(t))$	$\frac{1-q_i(t)}{\tau_i^2} - \frac{1}{2}q_i(t)x_i^2(t)$
	Utility	$\frac{\sqrt{2}}{\tau_i}\tan^{-1}\left(\frac{x_i\tau_i}{\sqrt{2}}\right)$
Reno (9)	$F_i(x_i(t), q_i(t))$	$\frac{1}{\tau_i^2} - \frac{1}{2}q_i(t)x_i^2(t)$
	Utility	$-\frac{2}{\tau_i^2 x_i}$
Reno (10)	$F_i(x_i(t), q_i(t))$	$\frac{1-q_i(t)}{\tau_i^2} - \frac{1}{2\tau_i}q_i(t)x_i(t)$
	Utility	$\frac{2}{\tau_i}\log(2 + x_i\tau_i)$
Vegas (15)	$F_i(x_i(t), q_i(t))$	$\frac{1}{(d_i \ + \ q_i(t))^2}\ \mathbf{sgn}\left(1 - \frac{x_i(t)q_i(t)}{\alpha_i d_i}\right)$
	Utility	$\alpha_i d_i \log x_i$

Table 1: Utility functions of TCP algorithms (Low, 2000)

3.5 AQM design

The result of the last subsection shows that the equilibrium of TCP/AQM is largely determined by the TCP algorithm (18)–(19) in that it alone defines the utility functions of the underlying optimization problem (1). This interpretation provides a convenient way to understand (the equilibrium properties of) various TCP algorithms in terms of their utility functions. The role of AQM (20)–(21) is to ensure complementary slackness condition (23) and to stabilize the equilibrium. This prompts the question: given a TCP algorithm (18)–(19), what is be "best" AQM?

This problem is formulated as an optimal control problem in (Kim and Low, 2002) as follows: the linearized TCP Reno (and queue) dynamics is treated as a dynamical system (F_i, G_i) and the AQM (20)–(21) is treated as the control input to this dynamical system. The control input (AQM scheme) is then chosen to minimize a linear quadratic cost of the transients in queue length, aggregate rate, jitter in the aggregate rate and price. By using a state-space model, AQM with PD (proportional-derivative) and PID (proportional-integral-derivative) structures are derived naturally as state feedback. Moreover, for the case of a single bottleneck link and N identical sources, the solution to the optimal control problem is a stabilizing AQM with a PID structure; conversely, any AQM with an appropriate structure solves the same optimal control problem with appropriate weighting matrices. Thus, whereas we can associate a utility function with any TCP algorithm, we can associate an optimal control problem with any AQM algorithm. Though the optimal AQM requires global information for its implementation, this model can potentially be useful to evaluate different practical AQMs as different approximations to the optimal AQM.

In this section, we have not dealt with the computational advantages of differential equation models in studying the performance of the Internet. The interested reader is referred to (Altman, Avrachenkov and Barakat, 2001; Bu and Towsley, 2001) for work along these lines.

4 Linearly Stable Protocols

So far, we have ignored the effect of network delay, i.e., we have assumed that the price information is instantly available at sources for their rate adjustments, and their new rates affect the link prices instantly. This is valid as we have been concerned with resource allocation *in equilibrium.* In this section, we study how delay determines the stability of the equilibrium point, and how to design TCP/AQM protocols that are linearly stable for general network topology, delay, and capacity.

4.1 Model with delay

We start by extending our dynamic model (16)–(21) to include network delay.

For each link l, the flow rate is determined by source rates that are τ_{li}^f time units earlier (compare with (16) and (17)):

$$y_l(t) = \sum_i R_{li} x_i(t - \tau_{li}^f) \tag{24}$$

where τ_{li}^f denote the equilibrium forward delays from sources to links. Similarly, the aggregate price observed at source i is

$$q_i(t) = \sum_l R_{li} p_l(t - \tau_{li}^b). \tag{25}$$

where τ_{li}^b are equilibrium backward delays in the feedback path. Let $\tau_i := \tau_{li}^f + \tau_{li}^b$ (for any link l in the path of source i) be the equilibrium round trip time of source i.

These equations can be represented in the Laplace domain in terms of the delayed forward and backward routing matrices:

$$[R_f(s)]_{li} \;=\; \begin{cases} e^{-\tau_{li}^f s} & \text{if } l \in i \\ 0 & \text{otherwise} \end{cases} , \qquad [R_b(s)]_{li} \;=\; \begin{cases} e^{-\tau_{li}^b s} & \text{if } l \in i \\ 0 & \text{otherwise} \end{cases} . \tag{26}$$

We have, in vector form,

$$y(s) \;=\; R_f(s) x(s) \tag{27}$$

$$q(s) \;=\; R_b(s)^T p(s). \tag{28}$$

Then a network of TCP/AQM is modeled by (18)–(21) and their interconnection is modeled by (24)–(25).

4.2 Stability of current protocols

Local stability of Reno/RED with feedback delays has been studied in (Hollot, Misra, Towsley and Gong, 2001a; Low, Paganini, Wang, Adlakha and Doyle, 2002) by linearizing (8) and (11)–(13). It is well known that TCP/RED can oscillate wildly and it is extremely hard to reduce the oscillation by tuning RED parameters, e.g., (May, Bonald and Bolot, 2000; Christiansen, Jeffay, Ott and Smith, 2000). The additive-increase-multiplicative-decrease (AIMD) strategy employed by TCP Reno (and its variants such as NewReno and SACK) and noise-like traffic that are not effectively controlled by TCP no doubt contribute to this oscillation. We show in (Low, Paganini, Wang, Adlakha and Doyle, 2002) however that these effects pale in comparison with protocol instability. The system is unstable when there is severe oscillation in aggregate quantities, such as queue length and aggregate window. We show that Reno/RED becomes unstable when delay increases, and more strikingly, when network capacity increases! Moreover, even if we smooth out AIMD, i.e., even if window is not adjusted on each acknowledgment arrival or loss event, but is adjusted periodically by the same *average* amount AIMD would over the same period, the oscillation persists. In particular, equation-based rate control will not help if the equation mimics the Reno dynamics. This suggests that Reno/RED is ill-suited for future networks where capacities will be large. We defer our discussion on the stability of Vegas/DropTail in the presence of delay to Section 4.5.

The above discussion motivates the design of simple, distributed algorithms that are scalable to general delay and capacity, as we now explain.

4.3 Primal algorithm

For ease of exposition, let us consider a single congestion controller accessing a single link with a utility function $U(x) = w \log(x)$, where w is some weighting factor. The primal algorithm (3) then becomes

$$\dot{x} = \kappa \left(w - x(t-\tau) p(x(t-\tau)) \right),$$

where τ is round-trip delay. The linearized version of this equation is

$$\delta\dot{x} = -\kappa (p(x^*) + x^* p'(x^*))\, \delta x(t-\tau) \tag{29}$$

where $\delta x(t)$ represents the perturbation from the equilibrium point x^* which satisfies

$$w - x^* p(x^*) = 0.$$

Taking the Laplace transform of (29) yields

$$(s + \kappa(p(x^*) + x^* p'(x^*)) e^{-s\tau}) \delta x(s) = \delta x(0)$$

where we have used the same notation δx to denote variable in both time and Laplace domains. Thus, the linear delay differential equation is stable if the roots of

$$s + \kappa(p(x^*) + x^* p'(x^*)) e^{-s\tau} = 0$$

do not lie in the right-half plane. From the Nyquist criterion, this would be true if the plot of

$$G(j\omega) = -\kappa\tau(p(x^*) + x^* p'(x^*))\frac{e^{-j\omega\tau}}{j\omega\tau}$$

as ω goes from $-\infty$ to $+\infty$ does not encircle the point -1 in the complex plane. It is easy to see that the values of ω for which $G(j\omega)$ intersects the real axis are given by $\cos(\omega\tau) = 0$, which implies that $\omega\tau = \pi/2 \pm 2\pi n$, $n = 0, 1, \ldots$. The maximum value of $sin(\omega\tau)/\omega\tau$ at these values for $\omega\tau$ is $2/\pi$. Thus, a little thought shows that the graph of $G(j\omega)$ always intersects the real axis to the right of -1 if

$$\kappa\tau(p(x^*) + x^* p'(x^*)) < \frac{\pi}{2}.$$

Roughly speaking, if the source chooses its controller gain κ inversely proportional to its round-trip delay, then the linearized version of the congestion controller is stable.

In (Johari and Tan, 2001), it was conjectured that, for general multi-link multi-source case, the network will be linearly stable under the primal algorithm (3) if, for each i,

$$\kappa_i\tau_i(q_i^* + \sum_{l\in i} y_l^* p_l'(y_l^*)) \quad < \quad \frac{\pi}{2} \tag{30}$$

where q_i^* is the aggregate price of source i in equilibrium and y_l^* is the aggregate arrival rate at link l in equilibrium. A weaker version of this conjecture was proved in (Massoulie, 2000) and the original conjecture was proved in (Vinnicombe, 2000; Vinnicombe, 2002), which also allows general utility functions. These results again lead to the following remarkable decentralized structure: each controller needs only to know its round-trip time and information about the price on its path to ensure the stability of the network of congestion controllers.

4.4 Dual algorithm

Recall the scaled gradient projection algorithm to solve the dual problem (4):

$$\begin{aligned} \dot{p}_l &= \gamma_l(y_l(t) - c_l) \\ x_i(t) &= U_i'^{-1}(q_i(t)) \end{aligned}$$

This dual algorithm is an example of the general TCP/AQM structure (18)–(21) with a static source algorithm and a first-order link algorithm. A special case with specific scaling γ_l and utility function U_i, developed in (Paganini, Doyle and Low, 2001), is the following TCP/AQM pair:

$$\dot{p}_l = \frac{1}{c_l}(y_l(t) - c_l) \tag{31}$$

$$x_i(t) = x_{m,i}\, e^{-\frac{\alpha_i q_i(t)}{M_i \tau_i}} \tag{32}$$

where $\alpha_i \in (0,1)$ and M_i is an upper bound on the number of bottleneck links in i's path. This source algorithm implies a utility function

$$U_i(x) = \frac{M_i \tau_i}{\alpha_i} x \left[1 - \log\left(\frac{x}{x_{m,i}} \right) \right], \quad \text{for } x \leq x_{m,i}.$$

The advantage of the dual algorithm (31)–(32) is that it automatically rescales itself with network delays and capacities to maintain linear stability. Specifically, suppose the routing matrix R has full row rank. Then there is a unique equilibrium rate and price vector (x^*, p^*). The linearized system around the equilibrium is described by (variables now denote perturbation):

$$\begin{aligned} \delta \dot{p}_l &= \frac{\delta y_l(t)}{c_l}, \quad \text{for all } l && (33) \\ \delta x_i &= -\frac{\alpha_i x_i^*}{M_i \tau_i} \delta q_i(t), \quad \text{for all } s && (34) \end{aligned}$$

where the source rates $x(t)$ and link prices $p(t)$ are interconnected by the delayed routing matrices defined in (24)–(25). Hence, the algorithm compensates for delay by scaling down the gain on source rates by their individual round trip times τ_i. It compensates for loop gain introduced by capacity and routing by scaling down the control gain at links by their capacities c_l and scaling it up at sources by their rates x_i^*. In other words, a source reacts more slowly if its round trip delay is large or if its rate is small; a link updates its price more slowly if it has a larger capacity. Note that network delay is the *only* open-loop parameter not under our control, and it *should* set the time-scale of the system response.

Then, it is proved in (Paganini et al., 2001) that, provided R has full row rank, the closed-loop system described by (33)–(34) and (24)–(25) is stable for *arbitrary* delays τ_i and link capacities c_l.

4.5 Primal-dual algorithms

The primal and dual algorithms of the last subsections are complementary in many ways. Both maintain linear stability in the presence of delays and for arbitrary link capacities. The primal algorithm uses a first order source algorithm and a static link algorithm whereas the dual algorithm is exactly the opposite. The primal algorithm allows general utility functions, and hence arbitrary fairness in rate allocation, but gives up tight control on utilization. The dual algorithm, on the other hand, can achieve very high utilization, but is restricted to a specific class of utility functions. By adding slow timescale dynamics at links (for primal algorithm) or at sources (for dual algorithm), it is possible to achieve both high utilization and arbitrary fairness, as we now explain.

The result of Section 4.3 can be further extended to suggest design choices when each link adapts its virtual queue capacity as in (7). It has been shown in (Kunniyur and Srikant, 2002b; Kunniyur and Srikant, 2002a) that choosing κ_i for each source i in a way that satisfies (30) and choosing the parameter α_l at each link inversely proportional to the longest source-destination round-trip time in the network ensures linear stability of the network of congestion controllers. In the case of a single link and single congestion-controlled source, the congestion control/pricing combination considered in (Kunniyur and Srikant, 2002b; Kunniyur and Srikant, 2002a) reduces to the following:

$$\begin{aligned} \dot{x} &= \kappa\,(w - x(t-\tau)p(x(t-\tau),\tilde{c})) \\ \dot{\tilde{c}} &= \alpha(c_l - x). \end{aligned}$$

In the above set of equations, we explicitly show the dependence of the marking function p on the virtual capacity $\tilde{c}$. In Section 4.3, considering only the dynamics at the sources, i.e., considering only the first of the above two equations and treating $\tilde{c}$ as a constant, we showed conditions on κ for the linearized version of the above system to be stable. The dynamics for $\tilde{c}$ allow for full utilization of the link. Note that the equilibrium point of the above equation is given by $x = c_l$, thus the link would be fully utilized if we can choose the parameters κ and α to ensure stability. To do this, in (Kunniyur and Srikant, 2002b; Kunniyur and Srikant, 2002a), the $\tilde{c}$ dynamics were thought of as a perturbation of the dynamics of the system without the $\tilde{c}$ dynamics and using results from perturbation theory, it was shown that κ and α can be chosen small enough to ensure linear stability. In the general result in (Kunniyur and Srikant, 2002b; Kunniyur and Srikant, 2002a), each congestion controller can choose κ knowing only pricing information in its route and its own round-trip delay, while each router can choose its α using a bound on the largest round-trip delay in the network. We also refer the reader to (Vinnicombe, 2002) for a robustness analysis of the above choice of design parameters.

While the primal algorithm needs to be augmented with slow timescale link dynamics to achieve full utilization, the dual algorithm of the last subsection has to be augmented with slow timescale dynamics at the source algorithm to track an arbitrary utility function. Consider the following adaptation of $x_{m,i}$ in (32):

$$\begin{aligned} x_i(t) &= x_{m,i}(t)\; e^{-\frac{\alpha_i q_i(t)}{M_i \tau_i}} \\ x_{m,i}(t) &= x_{\max,i}\; e^{\xi_i(t)} \\ \tau_i \dot{\xi}_i &= \beta_i(U_i'(x_i(t)) - q_i(t)) \end{aligned}$$

where the utility function U_i can be arbitrary. In equilibrium, $\dot{\xi}_i = 0$ and $U_i'(x_i^*) = q_i^*$, and hence the equilibrium rates x_i^* maximize aggregate utility with general utility functions U_i. The linearized source law is:

$$\tau_i \delta\dot{\xi}_i = \beta_i \left(U_i''(x_i^*)\delta x_i(t) - \delta q_i(t) \right) \tag{35}$$

$$\delta x_i(t) = x_i^* \left(\delta\xi_i(t) - \frac{\alpha_i}{M_i \tau_i} \delta q_i(t) \right) \tag{36}$$

Hence it replaces the static source law (34) by a lead-lag compensator that tracks arbitrary utility function at a slower timescale. It is shown in (Paganini, Wang, Low and Doyle, 2003) that, provided there is a common bound τ on delays, $\tau_i \leq \tau$ for all i, α_i and β_i can be chosen so that the linearized system (31) and (35)–(36) is stable for arbitrary capacities c_l and delays $\tau_i \leq \tau$.

We now discuss the stability of Vegas/DropTail and propose a modification that has a similar structure as (35)–(36).

In contrast to Reno and its variants, Vegas seems particularly well-suited for high speed networks. At high speed, Reno and its variants, with RED, become unstable as network capacity increases (Hollot et al., 2001a; Low, Paganini, Wang, Adlakha and Doyle, 2002). It also must maintain an exceedingly small loss probability in equilibrium that is difficult to reliably use for control. Vegas, on the other hand, scales correctly with capacity: compare (14) and (31). This built-in scaling with capacity makes Vegas potentially scalable to high bandwidth, in stark contrast to Reno and its variants. The source algorithm of Vegas, however, has a different scaling with respect to delay from those in (Paganini et al., 2001; Paganini et al., 2003).

To analyze its stability, note that

$$\operatorname{sgn}(z) \simeq \frac{2}{\pi} \tan^{-1}(\eta z)$$

The approximation becomes exact in the limit as $\eta \to \infty$. In (Choe and Low, 2003) the discontinuous Vegas algorithm (15) is approximated by the following differentiable function:

$$\dot{x}_i(t) = \frac{2}{\pi} \frac{1}{T_i^2(t)} \tan^{-1} \eta \left(1 - \frac{x_i(t) q_i(t)}{\alpha_i d_i} \right) \tag{37}$$

where again $T_i(t) = d_i + q_i(t)$ is the round trip time. Linearizing (14) and (37), Vegas is then modeled around its equilibrium, in Laplace domain, by

$$\delta x_r(s) = -\frac{x_r^*}{q_r^*} \frac{a_r}{sT_r + a_r} \delta q_r(s) \tag{38}$$

$$\delta p_l(s) = \frac{1}{c_l s} \delta y_l(s)$$

and the interconnection (26)–(28), where $a_r = 2\eta/\pi x_r^* T_r$. A sufficient stability condition is derived in (Choe and Low, 2003) that suggests that Vegas can become unstable at large delay, and this is confirmed by packet-level simulations.

To stabilize it, the following PD (proportional differential) modification to (37) is proposed in (Choe and Low, 2003):

$$\dot{x}_r = \frac{w}{T_r^2(t)} \cdot \tan^{-1} \eta_r(t) \left(1 - \frac{x_r(t) q_r(t)}{\alpha_r d_r} - \kappa_r(t) \dot{q}_r(t) \right)$$

where

$$\kappa_r(t) = \frac{1}{a} \cdot \frac{T_r(t)}{q_r(t)}, \qquad \eta_r(t) = \frac{\mu a}{w} \cdot x_r(t) T_r(t)$$

and μ and a are parameters to be chosen. The overall gain parameter η_r is proportional to the current window size: the larger the window, the more aggressive the response. The gain $\kappa_r(t)$ on the differential term is proportional to the ratio of round trip time to end-to-end queueing delay of source r, and serves as a normalization to $\dot{q}_r(t)$. The additional differential term $\kappa_r(t)\dot{q}_r(t)$ anticipates the future of $q_r(t)$. Without this term, source rate $x_r(t)$ will be increased if the number $x_r(t) q_r(t)$ of packets buffered in the links is small (compared with $\alpha_r d_r$). With this term, even when $x_r(t) q_r(t)$ is small, the source may decrease its rate if prices are rapidly growing, i.e. if $\dot{q}(t)$ is large. This modifies the linearized equation (38) into

$$\delta x_r(s) = -\frac{\mu x_r^*}{q_r^*} \left(\frac{s T_r + a}{s T_r + \mu a} \right) \delta q_r(s)$$

It is shown in (Choe and Low, 2003) how to choose the parameters μ, a to ensure the linear stability of the modified Vegas.

5 Modeling Assumptions

We have made several modeling assumptions in deriving congestion controllers and making parameter choices for these controllers. We comment on these assumptions in this section.

5.1 Deterministic fluid models

We have so far assumed that the network is described by a set of deterministic, delay-differential equations. It is more realistic to incorporate randomness in the network model (e.g. see (Baccelli and Hong, 2002)). A network such as the Internet is not only accessed by congestion-controlled sources, but is also used by sources that are either non-rate-adaptive or too short to be effectively controlled end-to-end. These uncontrolled or short-duration sources can be modeled as random disturbances to the deterministic system. It is well-known that file sizes in the Internet have a heavy-tailed distribution (see (Zhu, Yu and Doyle, 2001) for a plausible explanation of this phenomenon). For our purposes, this means that while most files have only a few packets, most packets belong to a few files. The small files are called *mice* and the huge files are called *elephants*. The aim of congestion control is to control the elephants to maximally utilize network bandwidth in a way that leaves the network queues mostly empty, so that mice can fly through the network with little delay or loss. Thus, the models in this paper address the control of elephants, but a more accurate model would include mice as random disturbances.

Recall the primal algorithm for the proportionally fair congestion controllers where the sources have a utility function $U(x) = \log(x)$:

$$\dot{x}_i = \kappa_i \left(1 - x_i(t-\tau)p\left(\sum_{j=1}^{n} x_j(t-\tau)\right)\right)$$

where we have assumed that there is a single link accessed by n sources, all with the same round-trip delay of τ. Suppose that, in addition to the congestion-controlled sources, there are n other sources with instantaneous rates $w_i(t)$, $i = 1, 2, \ldots, n$ that also share the link. Then, the congestion control equation becomes

$$\dot{x}_i = \kappa \left(1 - x_i(t-\tau)p\left(\sum_{j=1}^{n} x_j(t-\tau) + \sum_{j=1}^{n} w_j(t), n\tilde{c}\right)\right),$$

where we use the notation $p(x, \tilde{c})$ to denote the marking rate when the arrival rate is x and the virtual capacity is $\tilde{c}$. Defining $x(t) = \frac{1}{n}\sum_{i=1}^{n} x_i(t)$, and supposing that the marking function is such that $p(nx, n\tilde{c}) = p(x, \tilde{c})$, then the delay differential equation for the average rate $x(t)$ can be written as

$$\dot{x} = \kappa \left(1 - x(t-\tau)p\left(x(t-\tau) + \frac{1}{n}\sum_{j=1}^{n} w_j(t), \tilde{c}\right)\right).$$

If we make a law-of-large-numbers assumption on $\{w_j(t)\}$ whereby

$$\frac{1}{n}\sum_{j=1}^{n} w_j(t) \to 0$$

uniformly in $t \geq 0$, then it is reasonable to expect that the above differential equation will converge to the limit

$$\dot{x} = \kappa \left(1 - x(t-\tau)p(x(t-\tau), \tilde{c})\right)$$

The assumption $p(nx, n\tilde{c}) = p(x, \tilde{c})$ can be interpreted as that network capacity scales linearly with traffic demand, which is reasonable. For instance, the marking function

$$p(x, \tilde{c}) = \left(\frac{x}{\tilde{c}}\right)^B$$

can be thought of as the overflow probability in an $M/M/1$ queue with buffer size B, input rate x and service rate $\tilde{c}$. It seems reasonable that the capacity of the virtual queue $n\tilde{c}$ should be scaled with n, the number of sources. This would ensure that a constant amount of resource is used per source. Thus, when the virtual capacity is scaled with the number of sources, the assumption $p(nx, n\tilde{c}) = p(x, \tilde{c})$ is valid. This informal discussion is made precise in (Shakkottai and Srikant, 2001; Deb, Shakkottai and Srikant, 2002) taking into account many factors such as the fact that the transmission rate of each source should be non-negative, the marking could be based on queue length, there could be randomness in the marking process itself, the congestion controller could be the duality-based controller, etc.

5.2 Linearization

As we discussed in the previous subsection, the real network is described by a stochastic delay differential equation, but it seems reasonable to design controllers for such a system based on a deterministic delay differential equation model. However, even these models are difficult to analyze and therefore, for control parameter design, we resort to linearization. We now briefly discuss the effect of linearization and whether the conclusions from these linear models make sense for the nonlinear model.

We consider a single primal congestion controller accessing a single link, although similar results are possible for the dual controller as well:

$$\dot{x} = \kappa(1 - x(t-\tau)p(x(t-\tau))).$$

If the linearized form of the above equation is unstable, it does not necessarily mean that the trajectory of the nonlinear system will go to ∞. It is possible for the nonlinear system to be bounded, but oscillatory.

We will now argue that if $\kappa\tau$ is sufficiently small, the system will have bounded oscillations close to the equilibrium point. To do this, let M and l be some upper and lower bounds, respectively, on the trajectory of $x(t)$. To obtain a bound on M, we argue as follows: suppose $\dot{x}(t) < 0$ whenever $x(t) > M$, then we can conclude that $x(t)$ will be ultimately upper bounded by M. Similarly, if

$\dot{x}(t) > 0$ whenever $x(t) < l$, then we can conclude that $x(t)$ will be ultimately lower bounded by M. Now a straight forward argument from (Shakkottai, Srikant and Meyn, 2001) gives the following expressions for M and l:

$$\begin{aligned} M &= x^* + \kappa\tau \\ l &= x^* - \kappa\tau(1 - (\kappa\tau + x^*)p(\kappa\tau + x^*)). \end{aligned}$$

Thus, we can obtain an upper bound on $\kappa\tau$ for any desired upper bound on $M - l$. Conditions of the form $\kappa\tau < \eta$, for some constant η, seem to be fundamental in designing congestion controllers, both to guarantee linear stability and to reduce the magnitude of oscillation when the system is unstable. We also note that, if η is made sufficiently small, one can actually prove that the single source, single link, nonlinear delay differential equation is indeed stable (Deb and Srikant, 2001b).

6 Multicast

So far we have considered networks with *unicast* sessions only, i.e., each session has a single source and a single destination. On the other hand, there are many applications where a single source's transmission may be of interest to multiple destinations (also known as receivers). To model this scenario, let S denote the set of sessions and R_s be the set of receivers corresponding to a session $s \in S$. R_s is a singleton set for the unicast sessions and so we include the unicast case within the same framework. The receivers of a multicast group can be viewed as virtual sessions corresponding to that particular multicast session. Henceforth we use the terms virtual session and receiver interchangeably. We use the notation (s, r) to denote a virtual session corresponding to session s. Let L_{sr} be the set of all links in the route of a virtual session (s, r). Let S_l be the set of all sessions passing through link l and let V_{sl} be the set of all virtual sessions of the session s using link l. Let $U_{sr}(x)$ be the utility function of the virtual session (s, r). Also, associated with each link, let there be a function $p_l(x)$ which denotes the fraction of packets marked whenever there is congestion in that link when the total flow rate in that link is x. The details of how the marks are distributed among the virtual sessions sharing the link will be provided later. Let $x_{sr}(t)$ denote the rate at which the virtual session (s, r) sends data at time t. The total flow in link l at time t is given by $\sum_{s\in S_l}(\max_{(s,r)\in V_{sl}}(x_{sr}(t)))$.

The objective is to find rates so as to

$$\max_{x_{sr}\geq 0} \quad \sum_{s\in S}\sum_{r\in R_s} U_{sr}(x_{sr})$$

$$\text{subject to} \qquad \sum_{s \in S_l} \max_{(s,r) \in V_{sl}} x_{sr} \;\leq\; c_l, \quad \forall l \in L \tag{39}$$

The above maximization problem has a concave objective function and a convex constraint set and thus, has a unique optimal solution.

The non-differentiability of the max functions in (39) poses a challenge in studying the rate control equations and their stability. To bypass this difficulty, we approximate the function $\max_i(x_i)$ by the function $(\sum_i x_i^n)^{\frac{1}{n}}$, which uniformly converges to $\max_i(x_i)$ as $n \to \infty$ when all x_i's are non-negative real numbers. We hence consider the following optimization problem, for some sufficiently large n:

$$\begin{aligned} \max_{x_{sr} \geq 0} \qquad & \sum_{s \in S} \sum_{r \in R_s} U_{sr}(x_{sr}) \\ \text{subject to} \qquad & \sum_{s \in S_l} \Big(\sum_{(s,r) \in V_{sl}} x_{sr}^n \Big)^{\frac{1}{n}} \;\leq\; c_l, \quad \forall l \in L \end{aligned} \tag{40}$$

We now summarize the results from (Deb and Srikant, 2001b) where a set of rate control equations for the convex program (40) is presented and based on this, a heuristic for rate control for a network with multicast multirate sessions is developed.

Towards this end we consider a penalty function formulation of the convex program (40), in which we maximize the following function:

$$V_n(x) \;=\; \sum_{s \in S} \sum_{r \in R_s} U_{sr}(x_{sr}) - \sum_{l \in L} \int_0^{(\sum_{m \in S_l} (\sum_{(m,j) \in V_{ml}} x_{mj}^n)^{\frac{1}{n}})} p_l(u) du$$

As in the penalty function approach to the unicast congestion control problem, the following algorithm can be shown to solve the above problem:

$$\dot{x}_{sr} \;=\; \kappa_{sr} \left(1 - (U_{sr}'(x_{sr}))^{-1} \sum_{l \in L_{sr}} q_{l(sr)} \left(\sum_{m \in S_l} \max_{(m,j) \in V_{ml}} x_{mj} \right) \right) \tag{41}$$

Here the function $q_{l(sr)}(.)$ is link l's marking function for the virtual session (s, r). For all unicast sessions this function is the same as the original marking function $p_l(.)$. For all virtual sessions belonging to a multicast group the function $q_{l(sr)}(.)$ is same as $p_l(.)$ if that virtual session *alone* has the highest rate among all the virtual sessions of that multicast group through link l; $q_{l(sr)}(.)$ is equal to zero for any virtual session going through that link but having rate less than the multicast session rate through that link. If more than one virtual session has a rate equal to the multicast session rate through the link then for all those virtual sessions $q_{l(sr)}(.)$ is equal to $\frac{p_l(.)}{K}$, where K is the number of virtual sessions with rate equal to the multicast session rate. In practice, this can be viewed as being equal to $p_l(.)$ with probability $\frac{1}{K}$ and 0 with probability $(1 - \frac{1}{K})$.

This algorithm can be implemented as follows:

1. **Link algorithm**: Suppose link l has a marking function $p_l(.)$ which depends on the total flow rate through the link. Then it sets the ECN bit of a packet to 1 with probability $p_l(.)$, if the ECN bit is 0; if the ECN bit is 1 it is left as is. So packets from a session (multicast or unicast) are marked in proportion to their rates through the link.

2. **Node algorithm**: Every (multicast or unicast) session defines a directed tree on which packets flow from the source (root) towards the receivers (all leaves). Each packet is replicated k times at each node where k is the number of outgoing links from the node; call a node a *junction node* (with respect to a particular session) if $k > 1$. When a packet arrives at a non-junction node (with respect to the packet's session), it is forwarded to the (only) outgoing link as is. When a packet arrives at a junction node, the junction node chooses at random one of the receivers of this packet whose rate is the same as the session rate and sends the mark to that receiver. For all other receivers, the junction node unmarks any packet that is marked, before sending any copy of the packet.

3. **Receiver algorithm**: From the marked packets received each virtual session estimates the sum of the marking probabilities ($\sum_{l \in L_{sr}} q_{l(sr)}$) along its path. This can be achieved by counting the fraction of packets received with ECN bit set to 1 over some time interval. Then the rates are updated according to (41).

In the above congestion control mechanism, the only information the multicast flows require at the nodes (the junction nodes) is the identification of receivers whose rates are maximum.

The heuristic discussion here is rigorously proved in (Deb and Srikant, 2001a); see (Kar, Sarkar and Tassiulas, 2001) for related work. We have not discussed the extensions of the delay analysis or the adaptive virtual queue for multicast networks. We expect that analysis similar to the unicast case will also hold here.

7 Conclusions

We have presented a brief survey of some recent developments on modeling and design of congestion control protocols for the Internet. Interestingly, while there are two ways to solve the fair resource allocation problem for the Internet, namely, the primal and dual approaches, the choice of the congestion control parameters leads to design principles that are common to both approaches: the controller gains have to be chosen proportional to the inverse of the round-trip delays in the system, and if the congestion controller operates at a fast time scale, then the AQM scheme should operate at a slow time scale, and vice-versa.

Work is currently underway to translate these algorithms into practically implementable protocols that would allow the Internet to deliver faster service more reliably. On the theoretical front, much remains to be done, especially in understanding the implications of deterministic, linear modeling paradigm for the original highly nonlinear, stochastic system. It would also be important to develop models that show the connections between congestion control and other layers of the protocol stack such as routing, medium access, and call admission control, etc.

References

Adler, M., Cai, J.-Y., Shapiro, J. K. and Towsley, D. (2002). Estimation of congestion price using probabilistic packet marking. preprint.

Altman, E., Avrachenkov, K. and Barakat, C. (2001). TCP network calculus: The case of large delay-bandwidth product, *IEEE INFOCOM*.

Athuraliya, S., Li, V. H., Low, S. H. and Yin, Q. (2001). REM: active queue management, *IEEE Network* **15**(3): 48–53. Extended version in *Proceedings of ITC17*, Salvador, Brazil, September 2001. `http://netlab.caltech.edu`.

Athuraliya, S. and Low, S. H. (2000). Optimization flow control with Newton–like algorithm, *Journal of Telecommunication Systems* **15**(3/4): 345–358.

Baccelli, F. and Hong, D. (2002). AIMD, fairness and fractal scaling of TCP traffic, *Proceedings of IEEE Infocom*.

Brakmo, L. S. and Peterson, L. L. (1995). TCP Vegas: end-to-end congestion avoidance on a global Internet, *IEEE Journal on Selected Areas in Communications* **13**(8): 1465–80. `http://cs.princeton.edu/nsg/papers/jsac-vegas.ps`.

Bu, T. and Towsley, D. (2001). Fixed point approximation for TCP behavior in an AQM network, *Proceedings of ACM SIGMETRICS*.

Choe, H. and Low, S. H. (2003). Stabilized Vegas, *Proc. of IEEE Infocom*. http://netlab.caltech.edu.

Christiansen, M., Jeffay, K., Ott, D. and Smith, F. D. (2000). Tuning RED for web traffic, *Proceedings of ACM Sigcomm*.

Deb, S., Shakkottai, S. and Srikant, R. (2002). Stability and convergence of TCP-like congestion controllers in a many-flows regime. Technical Report, Univesity of Illinois. *To appear in the Proceedings of Infocom 2003.*

Deb, S. and Srikant, R. (2001a). Congestion control for fair resource allocation in networks with multicast flows, *Proceedings of the IEEE Conference on Decision and Control*.

Deb, S. and Srikant, R. (2001b). Global stability of congestion controllers for the Internet. Technical Report, Univesity of Illinois. To appear in the *Proceedings of the 2002 IEEE Conference on Decision and Control*.

Floyd, S. (1994). TCP and explicit congestion notification, *ACM Computer Communication Review* **24**: 10–23.

Floyd, S. and Jacobson, V. (1993). Random early detection gateways for congestion avoidance, *IEEE/ACM Trans. on Networking* **1**(4): 397–413. ftp://ftp.ee.lbl.gov/papers/early.ps.gz.

Gibbens, R. J. and Kelly, F. P. (1999). Resource pricing and the evolution of congestion control, *Automatica* **35**: 1969–1985.

Goel, A., Meyerson, A. and Bharghava, R. (2001). Using approximate majorization to characterize protocol fairness, *ACM SIGMETRICS*. Appeared as a short abstract. Full paper available at http://pollux.usc.edu/ agoel/research.html.

Hollot, C., Misra, V., Towsley, D. and Gong, W.-B. (2001a). A control theoretic analysis of RED, *Proceedings of IEEE Infocom*. http://www-net.cs.umass.edu/papers/papers.html.

Hollot, C., Misra, V., Towsley, D. and Gong, W.-B. (2001b). On designing improved controllers for AQM routers supporting TCP flows, *Proceedings of IEEE Infocom*. http://www-net.cs.umass.edu/papers/papers.html.

Jacobson, V. (1988). Congestion avoidance and control, *ACM Computer Communication Review* **18**: 314–329.

Johari, R. and Tan, D. (2001). End-to-end congestion control for the Internet: Delays and stability, *IEEE/ACM Transactions on Networking* **9**(6): 818–832.

Kar, K., Sarkar, S. and Tassiulas, L. (2001). A low-overhead rate control algorithm for maximizing aggregate receiver utility for multirate multicast sessions, *Proceedings of SPIE-ITCOM 2001.*

Kelly, F. (1997). Charging and rate control for elastic traffic, *European Transactions on Telecommunications* **8**: 33–37.

Kelly, F. (2001). Mathematical modelling of the Internet, *Mathematics Unlimited - 2001 and Beyond (Editors B. Engquist and W. Schmid)*, Springer-Verlag, Berlin, pp. 685–702.

Kelly, F. P., Maulloo, A. and Tan, D. (1998). Rate control for communication networks: Shadow prices, proportional fairness and stability, *Journal of Operations Research Society* **49**(3): 237–252.

Key, P., Kelly, F. and Zachary, S. (2000). Distributed admission control, *IEEE Journal on Selected Areas in Communications* **16**: 2617–2628.

Kim, K. B. and Low, S. H. (2002). Analysis and design of AQM for stabilizing TCP, *Technical Report caltechCSTR:2002.009*, Caltech.

Kunniyur, S. and Srikant, R. (2000). End-to-end congestion control: utility functions, random losses and ECN marks, *Proceedings of IEEE INFOCOM*, Tel Aviv, Israel.

Kunniyur, S. and Srikant, R. (2001). Analysis and design of an adaptive virtual queue algorithm for active queue management, *Proceedings of ACM Sigcomm*, San Diego, CA, pp. 123–134.

Kunniyur, S. and Srikant, R. (2002a). Designing AVQ parameters for a general topology network, *Proceedings of the Asian Control Conference*, Singapore.

Kunniyur, S. and Srikant, R. (2002b). Note on the stability of the AVQ scheme, *Proceedings of the Conference on Information Sciences and Systems*, Princeton, NJ.

Kunniyur, S. and Srikant, R. (2002c). A time-scale decomposition approach to adaptive ECN marking, *IEEE Transactions on Automatic Control* .

Lapsley, D. E. and Low, S. H. (1998). An IP Implementation of Optimization Flow Control, *Proceedings of the Globecom'98*.

Low, S. H. (2000). A duality model of TCP and queue management algorithms, *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management (updated version)*. `http://netlab.caltech.edu`. To appear in IEEE/ACM Transactions on Networking, 2003.

Low, S. H. and Lapsley, D. E. (1999). Optimization flow control, I: basic algorithm and convergence, *IEEE/ACM Transactions on Networking* **7**(6): 861–874. `http://netlab.caltech.edu`.

Low, S. H., Paganini, F. and Doyle, J. C. (2002). Internet congestion control, *IEEE Control Systems Magazine* **22**: 28–43.

Low, S. H., Paganini, F., Wang, J., Adlakha, S. A. and Doyle, J. C. (2002). Dynamics of TCP/RED and a scalable control, *Proc. of IEEE Infocom*. `http://netlab.caltech.edu`.

Low, S. H., Peterson, L. and Wang, L. (2002). Understanding Vegas: a duality model, *J. of ACM* **49**(2): 207–235. `http://netlab.caltech.edu`.

Massoulie, L. (2000). Stability of distributed congestion control with heterogenous feedback delays, *Technical Report, Microsoft Research, Cambridge, UK* .

Massoulie, L. and Roberts, J. (1999). Bandwidth sharing: objectives and algorithms, *Infocom'99*. `http://www.dmi.ens.fr/\%7Emistral/tcpworkshop.html`.

May, M., Bonald, T. and Bolot, J.-C. (2000). Analytic evaluation of RED performance, *Proceedings of IEEE Infocom*.

Mo, J. and Walrand, J. (2000). Fair end-to-end window-based congestion control, *IEEE/ACM Transactions on Netowrking* **8**(5): 556–567.

Paganini, F. (2002). A global stability result in network flow control, *Systems and Control Letters* . To appear.

Paganini, F., Doyle, J. C. and Low, S. H. (2001). Scalable laws for stable network congestion control, *Proceedings of Conference on Decision and Control*. `http://www.ee.ucla.edu/~paganini`.

Paganini, F., Wang, Z., Low, S. H. and Doyle, J. C. (2003). A new TCP/AQM for stability and performance in fast networks, *Proc. of IEEE Infocom*.

Ramakrishnan, K. K. and Floyd, S. (1999). A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481.

Shakkottai, S. and Srikant, R. (2001). Many-sources delay asymptotics with applications to priority queues, *Queueing Systems: Theory and Applications* . To appear.

Shakkottai, S., Srikant, R. and Meyn, S. (2001). Bounds on the throughput of congestion controllers in the presence of feedback delay, *Proceedings of the IEEE Conference on Decision and Control.*

Srikant, R. (2000). Control of communication networks, *in* T. Samad (ed.), *Perspectives in Control Engineering: Technologies, Applications and New Directions*, IEEE Press, pp. 462–488.

Stevens, W. (1999). *TCP/IP Illustrated: the Protocols*, Vol. 1, Addison-Wesley. 15th printing.

Vinnicombe, G. (2000). On the stability of end-to-end congestion control for the Internet, *Technical report*, Cambridge University, CUED/F-INFENG/TR.398.

Vinnicombe, G. (2002). On the stability of networks operating TCP-like congestion control, *Proc. of IFAC World Congress.*

Zhu, X., Yu, J. and Doyle, J. (2001). Heavy tails, generalized coding and optimal web layout, *Proceedings of IEEE INFOCOM.*