A Class of End-to-End Congestion Control Algorithms for the Internet

S. Jamaloddin Golestani
Bell Laboratories
Murray Hill NJ 07974
jamal@research.bell-labs.com

Supratik Bhattacharyya
Dept. of Computer Science
University of Massachusetts Amherst
Amherst MA 01002 USA
bhattach@cs.umass.edu

Abstract

We formulate end-to-end congestion control as a global optimization problem. Based on this formulation, a class of minimum cost flow control (MCFC) algorithms for adjusting session rates or window sizes are proposed. Significantly, we show that these algorithms can be implemented at the transport layer of an IP network and can provide certain fairness properties and user priority options without requiring non-FIFO switches. Two algorithm versions are discussed. A coarse version is geared towards implementation in the current Internet, relying on the end-to-end packet loss observations as indication of congestion. A more complete version anticipates an Internet where sessions can solicit explicit congestion information through a concise probing mechanism. We show that TCP congestion control, after some modification, may be treated as a special case of the MCFC algorithms.

1. Introduction

Congestion control in packet networks has proven to be a difficult problem, in general. However, this problem is particularly challenging in the Internet, due to very limited degrees of network observability and controllability. In order to accommodate rapid growth and proliferation, the design of the IP protocol and the requirements placed on individual subnetworks have been kept at a minimum. Consequently, the main form of congestion control possible in the current Internet is end-to-end control of user traffic at the transport layer. As exemplified by TCP [Jac88], this control must be exerted using only the limited network observation that sessions can locally make, based on their own performance. The prevalent form of service discipline in the Internet is FIFO queuing, and control approaches based on more sophisticated service disciplines are not easily applicable.

Although the current TCP congestion control has been relatively successful, its ability is exceedingly stretched by the rapid growth of the Internet and the proliferation of both real-time and multicast services. Over the past several years, considerable effort has been directed at improving the existing techniques of congestion control in the Internet and at introducing new approaches to accommodate the requirements of new services and applications [RJ88, CJ89, Flo91, Kes91, Mit92, MS90, MS93, ZDE$^+$93, Flo94, FJ93, FF96, Bra97].

In this paper, we formulate the end-to-end control of user traffic in IP networks as a global optimization problem. The optimization framework enables us to bring out, in a comprehensive and concrete manner, the tradeoff between avoiding congestion and satisfying users and the issues of fairness and priority among different users. Using this theoretical approach, we come up with a class of congestion control algorithms which have well-defined fairness properties and allow for the possibility of providing user priorities through the proper setting of certain design parameters associated with a particular user or application type.

Although methods of enforcing fairness or user priorities have been extensively studied in recent years, they are usually based on non-FIFO service scheduling at network switches where traffic streams meet and competitions arise [Zha91, Gol94]. Remarkably, in the class of algorithms presented here, we are able to achieve certain fairness properties and user priority options without requiring non-FIFO switches. The key to this success is the underlying optimization framework which strikes a balance between the satisfaction of various users and the congestion cost associated with various links.

Performing global optimization involving users and links scattered throughout the Internet is a seemingly infeasible task. Although it is well-known that network optimization problems can be solved using distributed computations, algorithms proposed for this purpose [Gal77, Gol79, GG80] have relied on the presence of sophisticated network layer protocols, a luxury not available in the Internet for end-to-end congestion control. A significant accomplishment of this paper is in showing how such optimization is indeed feasible in the Internet. We even show that the current TCP congestion control, after some modification, belongs to the class of optimal algorithms that we describe. We refer to these optimal algorithms, either collectively or individually, as the *minimum cost flow control* (MCFC) algorithm.

Two versions of the MCFC algorithm, referred to as the *coarse realization* and the *exact realization*, are explored in this paper. The coarse realization is geared towards implementation in the current Internet. This version of the algorithm, like TCP, relies on the end-to-end packet loss observations made by each session as indication of network congestion. The exact realization anticipates an Internet where sessions can solicit explicit congestion information from the network switches through a concise probing mechanism: some of the data packets of each session include a short congestion field which is modified by each switch that the packet visits.

Our systematic formulation of congestion control in the Internet provides a concrete framework to address several topics of increasing importance. One of these topics is congestion control for multicast communications. The theoretical framework developed here extends very naturally to multicast communications and allows for a systematic exploration of what is fair and how a particular notion of fairness impacts scalability. This subject is presented in a companion paper.

Another research topic of long standing is the comparison of algorithms that are based on explicit and implicit congestion notification. By implicit congestion notification we refer to the packet delay and loss observations that each session can locally make. The class of MCFC algorithms, by including both the coarse and the exact versions, provides a coherent setting to compare congestion control based on implicit and explicit notification.

The MCFC algorithm can be applied to both rate-based and window-based congestion control. Although both of these methods have been extensively studied, the real distinction between them and their dynamics are not always well-understood. Our approach to rate-based and window-based congestion control is founded on a distinction between quasi-static and dynamic fluctuations in network traffic. Therefore, before we embark on developing the optimization framework, we lay out in Section 2 our viewpoint on different time scales of congestion control and the corresponding roles that may be played by rate-based and window-based mechanisms.

2. Multiple Scales of Congestion Control

The dynamics of a network congestion control strategy can span multiple time scales. On the fastest time scale, congestion control should provide protection against sudden surges of traffic by quick reaction to buffer overloads. The reaction time in this type of control is, at best, in the order of one round-trip time since that is how fast news of congestion can reach a source node and the response to it propagate back to the trouble spot. We refer to this type of congestion control as *dynamic* and to the corresponding time scale as *short term*. On a slower time scale, congestion control could mean gradual but more steady reaction to the build-up of congestion, as perceived over a period involving tens or hundreds of round-trip times. It is on this time scale that notions such as the *average transmission rate* of a session, *rate allocation*, and *fairness* become meaningful. We shall use the terms *quasi-static* and *medium term* to refer to this type of congestion control and the corresponding time scale, respectively. Still, on slower time scales, congestion control can include longer term activities such as service scheduling on network switches and network reconfiguration, where possible. Our discussion in this paper is limited to the first two parts, i.e., dynamic and quasi-static congestion control.

Now, consider a window scheme for end-to-end congestion control. A window scheme keeps the amount of outstanding data for a given session limited to a maximum, called the window size. The window size may be changed in response to changing network conditions. But, let us first see what type of congestion control can be accomplished by window scheme if the window size is held constant. Consider a session s with a window size w_s. Assume an infinite source for the session, so that the window is always fully utilized. Denote by θ_s and ρ_s the round-trip time and the transmission rate of this session, averaged over some short term interval. According to Little's formula,

$$\rho_s \approx \frac{w_s}{\theta_s}. \tag{1}$$

Even with a fixed w_s, as the traffic increases and network links approach congestion, the round-trip time θ_s will increase, resulting in a proportionate reduction in the rate ρ_s. Notice that the reaction to increased queuing delay takes place within one round-trip time. Therefore, window scheme provides a form of dynamic congestion control even if the window size is not adjusted according to network conditions. If we now take the viewpoint of modifying the window size in response to *quasi-static* network conditions, then the window scheme combines dynamic and quasi-static congestion control.

The rest of this paper is concerned with developing algorithms for adaptive allocation of average session rates in the Internet, based on quasi-static network conditions. These allocated rates, once computed (e.g., by the sessions themselves), may be applied in two ways: directly by controlling the instantaneous transmission rate, or indirectly by a window scheme with the window size set in accordance with the allocated rate. Let r_s denote the average rate allocated to session s, based on algorithms to be discussed later. In the direct, i.e., rate-based approach, in order to apply r_s, the minimum spacing between packet transmission times may be either set to $1/r_s$, or more generally, determined through a leaky bucket mechanism. Alternatively, to apply r_s by means of window scheme, the window size w_s should be set to

$$w_s = r_s \cdot \tau_s, \tag{2}$$

where τ_s is the average round-trip time of session s.

The above approach combines the fast dynamics of window scheme with the quasi-static rate adjustments, provided that τ_s in (2) is the *medium term* average round-trip time of session s. To see why the *medium term* qualification in τ_s is necessary, let us denote by θ_s and ρ_s the round-trip time and the transmission rate of session s, averaged over some *short term* interval. It follows that:

$$\rho_s \approx \frac{w_s}{\theta_s} = r_s \frac{\tau_s}{\theta_s}, \tag{3}$$

clearly showing that with the increase of short term round-trip time θ_s, the short term rate ρ_s decreases, providing the quick reaction to congestion that was discussed. If we instead allow τ_s to be dominated by short term fluctuations of round-trip time, then we get $\tau_s \approx \theta_s$, and $\rho_s \approx r_s$, in effect neutralizing the dynamic component of window congestion control.

Consider using an exponentially weighted running average algorithm to update the estimation of τ_s, upon observing round-trip time $\theta^{(p)}$ of each new packet p from s:

$$\tau_s \longleftarrow (1-\beta)\tau_s + \beta \cdot \theta^{(p)}. \qquad (4)$$

If β is in the order of $1/w_s$, then the averaging interval in (4) is roughly one round-trip time (i.e., the time it takes to send w_s packets), which is not what we want. For averaging to take place over medium term, β in (4) should be at least one or two orders of magnitude smaller than $1/w_s$.

In Sections 3 the quasi-static allocation of session rates is formulated as a global optimization problem. While the solution algorithms are initially expressed in terms of the session rates r_s, we later use (2) to convert them to algorithms for directly updating the window sizes w_s.

3. A Global Optimization Framework

In this section, we formulate the allocation of session rates in a packet network as a global optimization problem. Formulation of network congestion control as a convex optimization problem was first studied in [Gol79, GG80]. However, in that study, routing and congestion control are treated as a unified problem. Consequently, congestion control parameters are determined based on and in conjunction with routing parameters, making the approach inapplicable to IP networks.

Consider a packet network, such as the Internet and denote the communication links (including both point-to-point and multiple access links) by integers $\ell = 1, \cdots, L$, and the network sessions by integers $s = 1, \cdots, S$. By a session, we refer to a one way flow of traffic between a given source and a given destination. Let r_s, denote the average rate of traffic of session s and f^ℓ, denote the average rate of traffic of link ℓ, all in bits/sec. The averaging intervals used in evaluating r_s and f^ℓ are medium term, i.e. they consist of tens or hundreds of round-trip times. We refer to r_s and f^ℓ, more briefly, as rate of session s, and *flow* of link ℓ, respectively. Denote the vector of session rates by $\vec{r} \stackrel{\triangle}{=} (r_1, r_2, \cdots, r_S)$, and the vector of link flows by $\vec{f} \stackrel{\triangle}{=} (f^1, f^2, \cdots, f^L)$.

We would like to come up with a quasi-static congestion control algorithm which determines and allocates the session rates r_s, based on the existing user demand and some reasonable notion of fairness. This goal can be accomplished by formulating network congestion control as a global optimization problem. We take the viewpoint of a supplier of network resources who can lose revenue either by failing to meet user demands or by causing long delays and lost messages due to congestion. Thus for each session s using the network we create a cost function $e_s(r_s)$, which is a decreasing function of the rate r_s allocated to s (Figure 1). As r_s is decreased, the cost in user dissatisfaction clearly increases. Similarly, for each communication link ℓ of the network, we create a cost function $g_\ell(f^\ell)$ (Figure 2), which is an increasing function of the flow f^ℓ on that link. As the flow approaches the capacity of the link, the average queue length of messages waiting to traverse the link increases and the danger of congestion goes up. Thus, $g_\ell(f^\ell)$ should represent the cost of this congestion danger.

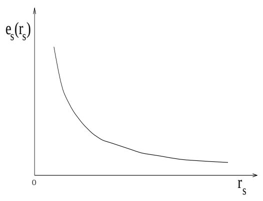

Figure 1. Cost of limiting the rate of session s to r_s.

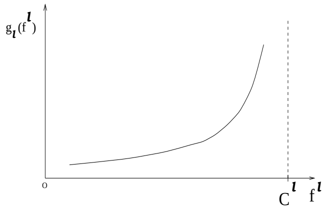

Figure 2. Congestion cost of the flow f^ℓ on link ℓ. C^ℓ is the link transmission speed.

The implications of different forms of the link and session cost functions will be discussed later. For the time being, we simply restrict them to be twice differentiable and convex, i.e., have nonnegative second derivative. We also assume that $e_s(r_s)$ and $g_\ell(f^\ell)$ are decreasing and increasing functions, respectively.

We distinguish between two types of possible routing in the network, *single-path* and *multi-path* routing. In *single-path* routing (such as the routing in the current Internet), the same path is applied to the traffic of each session, until the routing tables are changed. In *multi-path* routing, that we consider here for the sake of generality, a session's traffic may be divided over different paths leading to its destination. Of course, in both single-path and multi-path routing, the routing tables could be updated over the time. Unless otherwise stated, the discussions and conclusions of this paper apply to both single-path and multi-path routing.

Let ϕ_s^ℓ denote the fraction of traffic of s that is carried over link ℓ. Obviously, in single-path routing, ϕ_s^ℓ is one for each link ℓ along the path of s and zero, otherwise. In contrast, in multi-path routing, ϕ_s^ℓ may assume any value between zero and one. ¿From the above definitions it follows that,

$$f^\ell = \sum_{s=1}^{S} \phi_s^\ell \cdot r_s, \qquad \ell = 1, 2, \cdots, L, \qquad (5)$$

which simply says that the flow of each link is the sum of the rate of all sessions carried over it. Notice that if r_s and f^ℓ were taken to be short term averages, then (5) would not be quite correct, and in fact the right side minus the left side would represent the rate of buffer build up at link ℓ. Also, notice that in (5), packets lost at ℓ are assumed to be a negligible part of the total link traffic.

While, the routing parameters ϕ_s^ℓ show up in our formulation of congestion control and subsequent derivations, it turns out that the resulting end-to-end congestion control algorithm is realizable without requiring explicit knowledge about them. However, an assumption regarding the time scale of routing updates in the network is necessary for our formulation to be valid: the routing updates should take place over a time scale sufficiently longer than the time scale of congestion control updates so that the routing parameters can be assumed to be relatively constant.

Let r_s^d denote the average transmission rate desired by session s. The rate r_s to be allocated to each session s by the congestion control algorithm should satisfy:

$$0 \leq r_s \leq r_s^d. \tag{6}$$

As we shall see, an explicit knowledge of the desired rates r_s^d is not required by the rate allocation algorithms discussed in this paper. These parameters only appear in the interim mathematical steps leading to those algorithms.

We formulate network congestion control based on the following optimization problem expressed in terms of the rate vector $\vec{r}$ to be allocated to network sessions:

$$\min_{\vec{r}} J(\vec{r}) \stackrel{\triangle}{=} \sum_{s=1}^{S} e_s(r_s) + \sum_{\ell=1}^{L} g_\ell(f^\ell), \tag{7}$$

subject to constraint (6) and the expression of link flows in (5).

In order to state the optimality conditions for this problem, we introduce two new functions. First, we define the *incremental reward function* (or *reward function* for short) of each session s as

$$h_s(r_s) \stackrel{\triangle}{=} -e'_s(r_s), \quad s = 1, \cdots, S. \tag{8}$$

$h_s(r_s)$ is the incremental decrease in the cost of session s (due to user dissatisfaction) for a unit of increase in the allocated rate r_s, hence the name incremental reward function. Since $e_s(r_s)$ is by definition a decreasing convex function, $h_s(r_s)$ is positive and decreasing.

Next we define the *congestion measure* of a session s as the incremental cost of network congestion due to a unit of increase in r_s:

$$\gamma_s(\vec{f}) \stackrel{\triangle}{=} \frac{\partial}{\partial r_s} \sum_{\ell=1}^{L} g_\ell(f^\ell) = \sum_{\ell=1}^{L} \phi_s^\ell \cdot g'_\ell(f^\ell). \tag{9}$$

γ_s is denoted as a function of $\vec{f}$ to emphasize its dependence on the link flows in the network. Since by assumption, link cost functions are increasing and their derivatives are positive, a session's congestion measure is always a positive quantity. Clearly, with single-path routing, (9) reduces to

$$\gamma_s(\vec{f}) = \sum_{\ell \in \mathcal{P}_s} g'_\ell(f^\ell). \tag{10}$$

where $\mathcal{P}_s$ denotes the path of session s. We observe that with single-path routing (e.g., in the Internet), the congestion measure of each session s equals sum of the incremental costs of the links along its path.

Applying Kuhn-Tucker theory [Lue89] to the optimization problem (7), we are led to the following optimality conditions:

Theorem 1 *Assume that $g_\ell(.)$ and $e_s(.)$ have first and second derivatives satisfying $g'_\ell > 0, \quad g''_\ell > 0, \quad e'_s < 0,$ and $e''_s > 0$. The following is a set of necessary and sufficient conditions for the session rate vector $\vec{r}^*$ to minimize (7) subject to the constraint (5) and (6):*

$$h_s(r_s^*) \quad \begin{cases} \leq & \gamma_s(\vec{f*}), \quad \text{if } r_s^* = 0, \\ = & \gamma_s(\vec{f*}), \quad \text{if } 0 < r_s^* < r_s^d, \\ \geq & \gamma_s(\vec{f*}), \quad \text{if } r_s^* = r_s^d, \end{cases} \tag{11}$$

for $s = 1, \cdots, S$, where $\vec{f^*} = (f^{1*}, f^{2*}, \cdots, f^{L*})$ is the link flow vector corresponding to $\vec{r}^*$, as given by (5).

The interpretation of the above optimality condition is straight forward: at the optimal point $\vec{r}^*$, as long as constraint (6) is not active for a session s, the session's incremental reward function should be equal to the incremental cost of congestion, i.e., the session's congestion measure. If $r_s^* = 0$, ($r_s^* = r_s^d$), that is if r_s^* cannot be decreased (increased) anymore, then the session's incremental reward function can be smaller (larger) than the session's congestion measure.

3.1. Iterative Distributed Algorithms for the Solution

The constrained convex optimization problem (7) can be solved by means of a gradient projection algorithm with the following iterations:

$$r_s \longleftarrow \begin{cases} 0, \quad \text{if} \quad r_s + \mu \left(h_s(r_s) - \gamma_s(\vec{f})\right) \leq 0, \\ r_s^d, \quad \text{if} \quad r_s + \mu \left(h_s(r_s) - \gamma_s(\vec{f})\right) \geq r_s^d, \\ r_s + \mu \left(h_s(r_s) - \gamma_s(\vec{f})\right), \quad \text{otherwise.} \end{cases} \tag{12}$$

This algorithm converges to the optimal point of (7), provided that the step size μ is properly chosen [Lue89]. We shall refer to this algorithm or its variants as the *minimum cost flow control* (MCFC) algorithm.

Distributed execution of Iterations (12) by various sessions in the network is possible if, prior to each iteration, the current values of congestion measures γ_s are available. In Section 4, we will show how these congestion measures can be locally evaluated by the corresponding sessions, without involvement of the IP layer.

A priori knowledge of the desired session rates r_s^d is not actually necessary for the execution of the MCFC algorithm. When updating session rates, we can simply disregard the upper bounds r_s^d, and let the course of action determine whether or not a session s fully utilizes the allocated rate. In other words, we can replace Iteration (12) by

$$r_s \longleftarrow \max\left(0, \ \tilde{r}_s + \mu \left(h_s(\tilde{r}_s) - \gamma_s(\vec{f})\right)\right), \tag{13}$$

where r_s is the rate allocated to session s which may or may not be fully utilized, and $\tilde{r}_s$ is the average rate actually attained by s during the past iteration interval. In the rest of this paper, we simplify the presentation by ignoring the distinction between r_s and $\tilde{r}_s$, which reduces (13) to:

$$r_s \longleftarrow \max\left(0, r_s + \mu \left(h_s(r_s) - \gamma_s(\vec{f})\right)\right). \tag{14}$$

This simplification is equivalent to assuming that sessions are always greedy, i.e., they utilize whatever rate is allocated to them. The extension of subsequent results in the paper to the more general scenario is straight forward.

The speed of convergence of the MCFC algorithm can be significantly improved by incorporating the second derivatives of the cost function in Iteration (7) [Lue89, BG92], as follows:

$$r_s \longleftarrow \max\left(0,\; r_s + \mu \frac{h_s(r_s) - \gamma_s(\vec{f})}{\Gamma_s(\vec{f}) - h_s'(r_s)}\right), \qquad (15)$$

where

$$\Gamma_s(\vec{f}) \triangleq \frac{\partial^2}{\partial^2 r_s} \sum_{\ell=1}^{L} g_\ell(f^\ell) = \sum_{\ell=1}^{L} (\phi_s^\ell)^2 \cdot g_\ell''(f^\ell). \qquad (16)$$

In single-path routing, (16) reduces to

$$\Gamma_s(\vec{f}) = \sum_{\ell \in \mathcal{P}_s} g_\ell''(f^\ell). \qquad (17)$$

which facilitates the end-to-end evaluation of $\Gamma_s(\vec{f})$, as discussed later.

We defer a discussion of other considerations, such as asynchronous implementation of the MCFC algorithm and the impact of quasi-static traffic fluctuations to Section 4. In the next two subsections, we study the role of the session and link cost functions more closely, and identify some appropriate forms for them.

3.2. Fairness, Priorities, and the Impact of Session Reward Functions

In order to study the fairness properties of the session rate allocations at the optimal point, let us consider two sessions i and j, with identical cost functions and (therefore) identical reward functions, i.e., $e_i(r) = e_j(r)$, and $h_i(r) = h_j(r)$. Assume that the congestion measures seen by these two sessions at the optimal point are the same, i.e., $\gamma_i(\vec{f^*}) = \gamma_j(\vec{f^*})$. This means that sessions i and j induce the same increase in the total cost of network congestion for a unit of increase in their rate. This condition would apply if, for example, i and j share the same (set of) routes in the network. According to (11), as long as the desired rates of i and j permit, from $\gamma_i(\vec{f^*}) = \gamma_j(\vec{f^*})$, it follows that $r_i^* = r_j^*$; hence the following corollary:

Corollary 1 *To the extent permitted by session desired rates, at the optimal point of the MCFC algorithm, equal rates are allocated to sessions experiencing the same degree of network congestion (i.e., having the same congestion measures), unless the corresponding reward functions are different.*

In order to see how the rate allocated to a session may be influenced by the form of its reward function, we consider the class of reward functions:

$$h_s(r_s) = (\frac{\alpha_s}{r_s})^{\nu_s}, \qquad (18)$$

for some positive α_s and ν_s (curves a and b in Fig. 3). We can see from (18) that if the desired rate r_s^d is large enough,

$$r_s^* = \frac{\alpha_s}{\sqrt[\nu_s]{\gamma_s}} \;. \qquad (19)$$

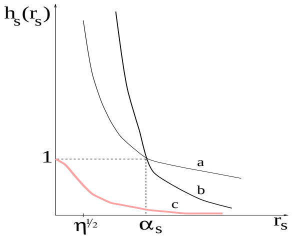

Figure 3. Comparison of different forms of a session's reward function : a and b correspond to (18) with $\nu_s = 1$ and $\nu_s = 3$ respectively. c illustrates the reward function (21) for $\nu_s = 2$, later shown to be associated with TCP.

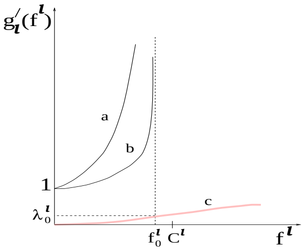

Figure 4. Comparison of different choices for the incremental congestion cost of a link : a and b correspond to (23) with $\nu = 1$ and $\nu = 0.33$, respectively. c shows $g_\ell'(f^\ell) = \lambda_\ell$, used in the coarse realization. Note that C^ℓ is the link transmission speed.

It follows that,

$$\frac{dr_s^*}{r_s^*} = -\frac{1}{\nu_s}\frac{d\gamma_s}{\gamma_s}. \qquad (20)$$

A few observation can be made from these results. First, we notice that the allocated rate is proportional to α_s. Therefore, a session with a large traffic volume, may be accommodated by assigning to it a large α_s. Next, we see in (19) that as congestion builds up in the network and γ_s increases, the allocated session rate decreases and the change is inversely proportional to $\sqrt[\nu_s]{\gamma_s}$. For $\nu_s = 1$, a doubling of the congestion measure γ_s, reduces the allocated rate by half. As ν_s becomes larger, the sensitivity of the allocated rate to the congestion measure goes down. For example, for $\nu_s = 4$, a doubling of the congestion measure cuts the allocated rate only by 16%. The role of ν_s is best illustrated by (20) which relates the changes in r_s^* and γ_s, in percentages terms.

We conclude that ν_s can be used as a priority assignment to sessions. Sessions with larger ν_s, would be cut less severely in response to network congestion. Similarly, a larger ν_s makes sessions less sensitive to the number of hops they must traverse in the network. We should realize, however, that this advantage is only relative. If all sessions are assigned a large ν_s, the congestion measures γ_s will increase enough until every body is cut back to the proper usage level, as discussed in Section 3.3.

As we will see later, realization of the MCFC algorithm in the *current* Internet is best accommodated by using session reward functions that are confined to an upper bound $h_{\max} \leq 1$. Keeping in mind that the reward functions must be positive and decreasing, we are led to consider the following class of reward functions, as another example:

$$h_s(r_s) = h_{\max}\frac{\eta_s}{\eta_s + r_s^{\nu_s}} \;, \qquad (21)$$

for some positive η_s and ν_s (curve c in Fig. 3). The priority implication of the index ν_s, earlier discussed in connection with (18), also applies to (21) as long as $r_s^* >> \eta_s^{1/\nu_s}$, where the functional form of (21) converges to that of (18). We will later show that the reward function (21), with $\nu_s = 2$ and $h_{\max} = 1$, is actually associated with the TCP-reno congestion control algorithm.

Notice that the fairness property stated in Corollary 1 applies regardless of the form of the reward function, provided that it is the same for the sessions in comparison.

3.3. Congestion Avoidance and the Impact of Link Cost Functions

The purpose of including link cost functions $g_\ell(f^\ell)$ in the optimization problem (7) is to inhibit the algorithm from driving the links into congestion by accepting too much traffic from the sessions. By proper choice of these cost functions, it is possible to keep the packet loss probability of each link below some desired maximum, as will be shown next. Rather than specifying the form of the cost functions $g_\ell(f^\ell)$, we carry out the discussion directly in terms of the incremental costs $g'_\ell(f^\ell)$, which are the actual quantities needed for the evaluation of congestion measures γ_s. We keep in mind that for $g_\ell(f^\ell)$ to be increasing and convex, $g'_\ell(f^\ell)$ must be positive and increasing.

Consider a link ℓ and denote the probability of packet losses due to buffer overflow over this link by λ^ℓ and the desired cap on this loss probability by $\lambda^\ell_\circ$. With a given buffer size and given traffic statistics, λ^ℓ is a monotonically increasing function of the link flow f^ℓ. Denote by $f^\ell_\circ$ the link flow at which the maximum permissible loss probability is reached, i.e., $\lambda^\ell(f^\ell_\circ) = \lambda^\ell_\circ$.

Theorem 2 *Consider a link ℓ and let $g'_\ell(f^\ell)$ be an arbitrary positive increasing function of f^ℓ over $[0, f^\ell_\circ)$, and $g'_\ell(f^\ell) = \infty$, for $f^\ell \geq f^\ell_\circ$. It follows that, at the optimal point of (7),*

$$\lambda^\ell(f^{\ell*}) < \lambda^\ell_\circ. \tag{22}$$

Therefore, once the algorithm converges to its optimal point, the loss probability of link ℓ is guaranteed to be below the desired cap $\lambda^\ell_\circ$. An analytically simple form for the incremental cost function of link ℓ is:

$$g'_\ell(f^\ell) = \frac{1}{(1 - f^\ell/f^\ell_\circ)^\nu} \ , \tag{23}$$

for some $\nu > 0$. As ν is decreased, $g'_\ell(f^\ell)$ becomes steeper (Fig. 4, curves a and b), which on the one hand, increases the link utilization at the optimal point and, on the other hand, reduces the speed of convergence.

So far, the incremental congestion cost of a link is specified as an explicit function of the link flow. Therefore, in the actual running of the algorithm, the link flow must be measured and plugged into the function $g'_\ell(f^\ell)$, to evaluate the incremental congestion cost. Alternatively, it is possible to use the average queue length of a link as the measurement parameter based on which the incremental congestion cost is specified. Let n^ℓ denote the average queue length of link ℓ. Denote by $n^\ell_\circ$, the average queue length corresponding to the flow $f^\ell_\circ$, i.e., the average queue length at which $\lambda^\ell = \lambda^\ell_\circ$. Consider specifying $g'_\ell(f^\ell)$ as an increasing function of n^ℓ which approaches infinity at $n^\ell = n^\ell_\circ$. For example, let:

$$g'_\ell(f^\ell) \triangleq \frac{1}{(1 - n^\ell/n^\ell_\circ)^\nu} \ . \tag{24}$$

It is easy to verify that $g'_\ell(f^\ell)$, as implicitly defined in the above, satisfies the properties required by Theorem 2.

It must be noted that the strong congestion avoidance property stated in Theorem 2, hinges on the ability to specify the threshold parameters $f^\ell_\circ$ in (23) or $n^\ell_\circ$ in (24), based on the desired loss probability cap $\lambda^\ell_\circ$. Obviously, the relationship between these parameters depends on the statistics of the traffic passing through the link, which is not easily predictable. Therefore, the threshold parameter of choice, i.e., $f^\ell_\circ$ or $n^\ell_\circ$, must be specified in anticipation of likely changes in traffic statistics, such as burstiness. A main distinction between defining the incremental congestion cost directly in terms of f^ℓ, or implicitly in terms of n^ℓ, is in the sensitivity of the corresponding threshold parameter to the traffic statistics. Intuitively, it seems that $n^\ell_\circ$ should be less sensitive than $f^\ell_\circ$ to changes in traffic statistics, suggesting that the incremental congestion cost should be specified in terms of the average queue length. More research is needed to conclusively determine the best measurement parameter(s) to be used for specifying the incremental congestion cost of a link.

4. Realization of the MCFC algorithm in IP Networks

The main difficulty facing the realization of the MCFC algorithm (14) is the distributed computation of the congestion measures γ_s. In a network with a highly developed network layer, the task of computing congestion measures and distributing them to the corresponding sessions (or access points) can be performed by a specially designed network layer protocol, in possible cooperation with the routing protocol. In the Internet or other IP networks, realization of the MCFC algorithm is more challenging since it should be done without explicit knowledge of the routing parameters and without expecting cooperation from the IP layer.

In this section, we introduce two possible realization for the MCFC algorithm at the transport layer of an IP network: an exact realization requiring modest cooperation by network switches, and a coarse realization with no such requirement, which is therefore applicable to the current Internet. We have also come up with a hybrid realization of the algorithm in a network consisting of both cooperative and non-cooperative switches, facilitating transition from the coarse to the exact realization when the necessary protocol and switch enhancements are gradually introduced. Due to space limitation, this hybrid realization is not discussed here.

4.1. Exact Realization with Switch Cooperation

Distributed execution of the MCFC algorithm (14) by various network sessions is possible if the sessions have a way of evaluating the corresponding congestion measures.

There are two basic requirements for the evaluation of congestion measures γ_s by each session s. First, at each link ℓ, there must be a local capability to evaluate the incremental congestion cost $g'_\ell(f^\ell)$, on an ongoing basis. Second, there must be a way of communicating this information to the sessions traversing link ℓ. The method we employ to relay congestion information to the sessions is both simple and concise. But more significantly, it relaxes the need for explicit knowledge about routing parameters, thereby enabling a realization of the algorithm in the Internet.

Consider a packet network with the following capabilities:

1. Each switch (or router) in the network has the capability of estimating $g'_\ell(f^\ell)$, for each link originating from it. This estimation is performed on an on-going basis. Here, the term switch refers to any multiplexing point in the network.

2. Some of the data packets traversing the network are marked by the source (or the access point) as *probe* packets. Each *probe* packet, in addition to user data, includes a short *congestion field* to carry congestion information. This field is initially set to zero, at the source.

3. Each switch in the network, before forwarding a probe packet over a link ℓ, increments its congestion field by the current estimate of the link's incremental cost $g'_\ell(f^\ell)$.

Theorem 3 *Consider a session s and a probe packet p belonging to this session. Let $\gamma^{(p)}$ denote the value of the congestion field of p upon arrival to the destination. Then,*

$$E\{\gamma^{(p)}\} = \gamma_s. \tag{25}$$

For single-path routing, (25) reduces to

$$\gamma^{(p)} = \gamma_s. \tag{26}$$

Here, we present an intuitive explanation about this theorem. Consider a short period of time during which the statistics of network traffic and the routing parameters do not change. Let n probe packets from a session s be transmitted during this interval. Since ϕ_s^ℓ is the fraction of packets of s which use ℓ, if n is sufficiently large, about $\phi_s^\ell \cdot n$ of these probe packets traverse ℓ, each of which will have its congestion field incremented by $g'_\ell(f^\ell)$. The total increase of the congestion field of probe packets which traverse ℓ will be $\phi_s^\ell \cdot n \cdot g'_\ell(f^\ell)$. Accordingly, the total value of the congestion field of all of the n probe packets upon arrival to the destination will be $\sum_{\ell=1}^{L} \phi_s^\ell \cdot n \cdot g'_\ell(f^\ell) = n \cdot \gamma_s$. Therefore, γ_s equals the average of the congestion field of the probe packets. Notice that with single-path routing, the path traveled by p and the value of $\gamma^{(p)}$ are deterministic, simplifying (25) to (26).

As Theorem 3 shows, with single-path routing, the value of a session's congestion measure at any given time can be obtained from a single probe packet. It is interesting to see whether the second order congestion parameters Γ_s required to implement the second order algorithm (15) can also be determined in a similar manner. For single-path routing, in view of the similarity of (10) and (17), Γ_s can be determined based on an identical approach; it suffices to designate a new field in each probe packet to second order information and have this field incremented by each visited switch, in a similar fashion. For multi-path routing, however, it can be shown that there is no way to implement the second order algorithm (15), short of full cooperation by the network layer. In the Internet, since the routing is single-path, both the first and the second order algorithms are realizable. For the rest of this paper, we limit our attention to the first order algorithm (14).

We now consider the important issue of interactions between the quasi-static changes in network traffic and the algorithmic iterations in (14). Ideally, one would like to see the network traffic remain stationary until the algorithm converges to its optimal point. In real network operation, however, due to quasi-static traffic changes, the optimal point is not stationary and may be viewed as a moving target that the algorithm tries to reach. Although this target may not be reached exactly, with a sufficient speed of convergence, the algorithm should be able to keep up with the pace of network changes and follow the optimal point relatively closely. Since the network traffic is an aggregation of traffic from many sources, its changes are typically slower than the dynamics of individual sessions [BG92].

In general, a distributed algorithm such as (14) may be executed either synchronously, or asynchronously [BT89]. In a loosely connected network such as the Internet, synchronous execution of (14) by various sessions is not feasible. Moreover, the potential benefit of synchronous execution in terms of providing faster convergence is either minimized or totally removed by the quasi-static traffic variations.

In an asynchronous implementation of (14), each session updates its input rate without timing coordination with other sessions. To increase the speed of convergence, the session congestion measures should be updated regularly, based on regular transmission of probe packets. Similarly, each link should update its incremental congestion cost on a regular basis. Evaluation of session congestion measures and link incremental costs should involve a limited memory span, so that the information regarding past network status is slowly forgotten and replaced by the more recent network conditions. This goal may be accomplished by updating session congestion measures and link average queue lengths, using the following exponentially weighted running averages:

$$\gamma_s \longleftarrow (1-\beta_s)\gamma_s + \beta_s \cdot \gamma^{(p)}, \tag{27}$$

and

$$n^\ell \longleftarrow (1-\beta^\ell)n^\ell + \beta^\ell \cdot n_t^\ell, \tag{28}$$

where $\gamma^{(p)}$ is the congestion field of the received probe packet, and n_t^ℓ is the queue length at the time of update.

The time constant in the above averaging algorithms (i.e. the memory span, measured in seconds) is equal to $1/\beta_s$ (or $1/\beta^\ell$) times the corresponding updating interval. The choice of this time constant involves a trade-off between accurately measuring traffic conditions in the network and quickly responding to it. Conceptually, it seems desirable to apply the same time constant to the evaluation of link incremental costs, throughout the network. However, due to the wide range of link and session transmission rates in a diverse network such as the Internet, it may prove inevitable to apply different time constants to various parts of the network.

Once a session's congestion measure is evaluated, its rate can be updated through

$$r_s \longleftarrow \max\left(r_s^{\text{init}}, r_s + \mu\left(h_s(r_s) - \gamma_s(\vec{f})\right)\right), \qquad (29)$$

where r_s^{init} is a small rate initially allocated to each new session s to enable transmission of probe packets needed for the initial evaluation of congestion measure. Notice that a session need not execute (27) and (29) with the same frequency. The congestion measure is updated each time a new probe packet is received, while the rate may be updated at the same time, or less frequently.

An alternative to explicitly updating the congestion measure through (27) and using it for rate updates, is to update the rate directly based on the congestion field of the received probe packets p:

$$r_s \longleftarrow \max\left(r_s^{\text{init}},\ r_s + \epsilon\left(h_s(r_s) - \gamma^{(p)}\right)\right). \qquad (30)$$

One can easily verify that the statistical average of the rate change in (30) is identical to the rate change according to (29), provided that the right step size ϵ is used. Although in this approach, the congestion measure is not explicitly determined, updating the rate through (30) amounts to maintaining an implicit estimation of the congestion measure.

A session's rate or congestion measure may be updated by the source or receiver (or by a policing entity, where such an entity exists). Obviously, each approach has different implications on the design of transport protocols, the control information which must be exchanged between the source and receiver, and the interaction between error control and congestion control. These issues fall beyond the scope of the present paper.

4.2. Coarse Realization in the Current Internet

In this section, we develop a realization for the MCFC algorithm without using probe packets and requiring explicit congestion information from network switches.

In the absence of explicit congestion notification, the only observation a session can have about the network is through its own performance, i.e., the loss and delay of its own packets. We try to choose a form for the cost functions $g^\ell(f^\ell)$ so that the resulting congestion measures γ_s can be best estimated through the available loss and delay information.

Let us denote the end-to-end loss probability and the *average* delay of packets of session s, by λ_s, and D_s, respectively. Similarly, we denote the average delay of each link ℓ, by D^ℓ.

Theorem 4 *The loss probability and the average delay of each session s can be expressed as,*

$$D_s(\vec{f}) = \sum_{\ell=1}^{L} \phi_s^\ell \cdot D^\ell(f^\ell), \qquad (31)$$

and

$$\lambda_s(\vec{f}) \approx \sum_{\ell=1}^{L} \phi_s^\ell \cdot \lambda^\ell(f^\ell), \qquad (32)$$

where (32) is valid assuming that all sessions sharing a link ℓ encounter the same loss probability at that link. The approximation in (32) is good for $\lambda_s << 1$.

By comparing (31) and (32) with (9), we get the following corollary:

Corollary 2 *Consider the following incremental congestion costs for the links of the network:*

$$g'_\ell(f^\ell) \triangleq \xi \cdot D^\ell(f^\ell) + \lambda^\ell(f^\ell), \qquad \ell = 1, 2, \cdots, L. \quad (33)$$

The congestion measure of each session s can be stated as:

$$\gamma_s(\vec{f}) \approx \xi \cdot D_s(\vec{f}) + \cdot\lambda_s(\vec{f}). \qquad (34)$$

In principle, a session can estimate the average delay and loss probability associated with its own transmissions. Therefore, Corollary 2 suggests that if incremental cost functions of the form (33) are a suitable representation for the level of congestion on individual links, then the associated congestion measures can be estimated locally by the sessions, without receiving explicit congestion notification from the switches. The cost function specified in (33) meets the convexity requirement since $D^\ell(f^\ell)$ and $\lambda^\ell(f^\ell)$ are both increasing functions of f^ℓ. To see how well it can indicate congestion, we consider the delay and loss terms in (33), separately. While there is a positive correlation between the average delay and the level of congestion on a link, average delay is not indicative of congestion, in itself. Other information, such as the propagation delay and the available buffer space (or the acceptable range of queuing delays) are essential to infer the level of congestion associated with a given average delay. In contrast, the loss probability provides a more conclusive indication of the severity of congestion, suggesting that we should use the second term in (33), and set $\xi = 0$:

$$g'_\ell(f^\ell) \triangleq \lambda^\ell(f^\ell), \qquad (35)$$

which gives rise to the following congestion measure:

$$\gamma_s(\vec{f}) = \lambda_s(\vec{f}). \qquad (36)$$

Note however that if a large fraction of losses are due to transmission error, as could be the case in wireless communications, link loss probability cannot be trusted as a good indicator of congestion, either.

The strong congestion avoidance property stated in Theorem 2 was based on link cost functions that approach infinity as the link flow exceeds a critical threshold, and does not apply with link cost functions chosen as in (35). In fact, it is easy to see that if link cost functions (35) are used in conjunction with unbounded session reward functions such as (18), the MCFC algorithm could drive the network into heavy congestion. If, on the other hand, the reward functions are appropriately bounded, small loss probabilities can still be guaranteed at the optimal point of the algorithm, as established by the following theorem:

Theorem 5 *Consider a network with single-path routing and let the following bound be satisfied by all session reward functions:*

$$h_s(0) \leq h_{\max}, \qquad s = 1, 2, \cdots, S, \tag{37}$$

for some $h_{\max} \leq 1$. It follows that, at the optimal point of the MCFC algorithm, for all sessions s and links ℓ, $\lambda_s < h_{\max}$, and $\lambda_\ell < h_{\max}$, provided that the initial session rates r_s^{init} are sufficiently small and, by themselves, do not lead to excessive loss.

What is ignored by the above theorem, is the difficulties and inaccuracies involved in the estimation of congestion measures $\gamma_s = \lambda_s$, an issue that we now explore. To make an analogy with the estimation of congestion measures using probe packets, we might associate a parameter $\gamma^{(p)}$ with each packet p, and assume that $\gamma^{(p)} = 1$, if the packet is lost, and $\gamma^{(p)} = 0$, otherwise. With this convention, we notice that,

$$\gamma_s = \lambda_s = E\{\gamma^{(p)}\}, \tag{38}$$

which parallels (25) in Theorem 3. For the asynchronous implementation of the MCFC algorithm, we may again estimate γ_s using the exponentially weighted running average algorithm (27). In the present case, (27) may be restated as the following iteration, executed every time a new loss or successful transmission is observed:

$$\gamma_s \longleftarrow \begin{cases} (1-\beta)\gamma_s, & \text{successful transmission,} \\ (1-\beta)\gamma_s + \beta, & \text{packet loss.} \end{cases} \tag{39}$$

The algorithmic similarities between estimating γ_s in the coarse and exact realizations, should not obscure a fundamental difference between the two cases regarding the range of statistical fluctuations in $\gamma^{(p)}$ and the accuracy of estimations. We notice from Theorem 3 that in the exact realization in a network with single-path routing, one probe packet is enough to determine the congestion measure. In the coarse realization, on the other hand, the analogous parameter $\gamma^{(p)}$ associated with each packet p, is either one or zero, with an average typically in the order of few percent or less. Here, due to the random nature of $\gamma^{(p)}$, a much larger number of observations are necessary before algorithm (39) converges to a reasonable estimation of the end-to-end loss probability. As a numerical example, if $\lambda_s = 0.01$, typically one out of every 100 packets are lost, implying that at least several hundred observations are needed for a meaningful estimation of λ_s. This sharp difference with the exact realization is the result of restricting information about network status to the packet losses locally observed.

We should emphasize that the choice of link cost functions in (33) or (35) was dictated by the requirement of coming up with congestion measures γ_s that sessions can locally evaluate, using their own loss and delay observations. It is possible to show that, in a network of arbitrary topology, no other form of link cost functions can satisfy this requirement.

One way to run the coarse MCFC algorithm is to update the rate via (29), based on explicit estimation of γ_s obtained in (39). An alternative approach, like in the exact realization, is to directly update the rate, upon observing each new loss or successful transmission, by way of (30). In the coarse realization, due to the wide random fluctuations of $\gamma^{(p)}$, (30) constitutes a stochastic gradient algorithm. The choice of the step size ϵ in this case involves hard tradeoffs, as will be illustrated in the simulation section. A small ϵ, prolongs the time necessary for the rate of new sessions to reach the final value. A large ϵ, on the other hand, gives rise to large oscillations in the session rates, induced by the random fluctuations of $\gamma^{(p)}$. This difficulty can be overcome by adopting a variable step size in (30), i.e. adjusting ϵ as a function of iteration number, session rate, or some other parameter.

For the coarse realization, we restate (30) as:

$$r_s \longleftarrow \begin{cases} r_s + a_s(r_s), & \text{successful trans.} \\ \max\left(r_s^{\text{init}},\ r_s - b_s(r_s)\right), & \text{packet loss,} \end{cases} \tag{40}$$

where

$$a_s(r_s) = \epsilon_s(r_s) \cdot h_s(r_s), \tag{41}$$

and

$$b_s(r_s) = \epsilon_s(r_s)\left(1 - h_s(r_s)\right). \tag{42}$$

In the above equations, we have denoted ϵ_s as a function of r_s, in order to emphasize the possibility of changing the step size during the course of the algorithm, based on the value attained by r_s, (or some other criteria). According to (40), a session's rate must be increased by $a_s(r_s)$, each time a packet is successfully transmitted, and reduced by $b_s(r_s)$, each time a packet loss is observed.

As discussed in Section 2, the allocated rate r_s may be enforced by means of the window scheme, with the added benefit of combining fast dynamics of the window scheme with the quasi-static control that the MCFC algorithm provides. Using (2) and (40), we get the following iteration for directly updating the window size:

$$w_s \longleftarrow \begin{cases} w_s + A_s(w_s), & \text{successful trans.} \\ \max\left(w_s^{\text{init}},\ w_s - B_s(w_s)\right), & \text{packet loss,} \end{cases} \tag{43}$$

where

$$A_s(w_s) = \tau_s \cdot a_s(\frac{w_s}{\tau_s}), \tag{44}$$

$$B_s(w_s) = \tau_s \cdot b_s(w_s \cdot \tau_s), \tag{45}$$

and

$$w_s^{\text{init}} = \tau_s \cdot r_s^{\text{init}}. \tag{46}$$

Obviously, the smallest feasible value for the initial window w_s^{init} is the size of one packet.

The window version of the MCFC algorithm in (43), reveals significant similarity to TCP congestion control which also decreases the window size in reaction to packet losses and increases it when packets are successfully transmitted. In Section 5.1, we further explore the relationship between TCP congestion control and the MCFC algorithm.

5. Comparison with Alternative Schemes

In this section, we provide a comparison between the MCFC algorithm and some of the congestion control schemes previously proposed for the Internet. These schemes are the TCP congestion control [Jac88] currently used in the Internet, the Binary Feedback Scheme [RJ88], the Random Early Detection Gateways [FJ93], and the Dynamic Adaptive Windows [Mit92, MS90, MS93]. While the global optimization framework is a foundation unique to the MCFC algorithm, there are important commonalities between the above schemes and the MCFC algorithm, regarding the underlying ideas or methods of execution. These common features allow us to apply some of the insight gained from the design and analysis of the MCFC algorithm to other schemes and develop a clearer understanding of the merits and drawbacks of each approach.

5.1. TCP Congestion Control

In order to compare the MCFC algorithm with TCP congestion control, we consider the window-based coarse realization of the algorithm with $a_s(r_s)$ and $b_s(r_s)$ selected as:

$$a_s(r_s) = \zeta \frac{\eta}{r_s} \ , \tag{47}$$

and

$$b_s(r_s) = \zeta \cdot r_s \ , \tag{48}$$

which, in view of (41) and (42), correspond to the reward function

$$h_s(r_s) = \frac{a_s(r_s)}{a_s(r_s) + b_s(r_s)} = \frac{\eta}{\eta + r_s^2}, \tag{49}$$

which is a special case of (21) for $\nu_s = 2$, $h_{\max} = 1$, and $\eta_s = \eta$ (curve c in Fig. 3). Notice that ζ has no effect on the form of the reward function and merely determines the step size of the algorithm. In the window-based implementation, in view of (44) and (45), window sizes may be updated using iteration (43) with:

$$A_s(w_s) = \zeta \frac{\eta \cdot \tau_s^2}{w_s}, \tag{50}$$

and

$$B_s(w_s) = \zeta \cdot w_s. \tag{51}$$

We will refer to the above coarse MCFC algorithm as the *modified TCP* algorithm. To see the reason behind this naming, consider the function

$$A'_s(w_s) = \zeta \cdot \frac{\eta'}{w_s} \ , \tag{52}$$

which differs from $A_s(w_s)$ in the missing term τ_s^2. It is easy to see that the adjustment of window sizes in TCP-reno, after the slow start phase, can be expressed by (43), using $A'_s(w_s)$ in place of $A_s(w_s)$ and applying coefficients $\eta' = 2$ and $\zeta = 0.5$. It turns out that the essential difference between the special case of the MCFC algorithm, here called modified TCP, and TCP-reno is the extra term τ_s^2, in (50).

In order to determine the impact of the extra term τ_s^2 in the modified TCP algorithm, let us compute the statistical average of the window size change during one iteration of (43):

$$\begin{aligned} E\{\Delta w_s\} &= \lambda_s \cdot E\{\Delta w_s | \text{ packet loss}\} \ + \\ &\quad (1-\lambda_s) \cdot E\{\Delta w_s | \text{ successful transmission}\} \\ &= -\lambda_s \cdot B_s(w_s) + (1-\lambda_s) \cdot A_s(w_s) \\ &= A_s(w_s) - \lambda_s(A_s(w_s) + B_s(w_s)). \end{aligned} \tag{53}$$

Assuming that the algorithm reaches the optimal point $\vec{w} = (w_1^*, w_2^*, \cdots, w_S^*)$, the statistical average of change at this point should be zero. It follows that

$$\frac{A_s(w_s^*)}{A_s(w_s^*) + B_s(w_s^*)} = \lambda_s. \tag{54}$$

For the modified TCP algorithm, we conclude from (50) and (51) that

$$\frac{\eta}{\eta + (w_s^*/\tau_s)^2} = \lambda_s. \tag{55}$$

Notice that the LHS of (55) is equal to $h_s(r_s^*)$, reaffirming the optimality condition in (11). For TCP-reno, on the other hand, by substituting $A'_s(w_s^*)$ for $A_s(w_s^*)$ in (54) and applying (51) and (52), we get

$$\frac{\eta'}{\eta' + w_s^{*2}} = \lambda_s. \tag{56}$$

The impact of the term τ_s^2 in the modified TCP algorithm is now clearly explained by comparing (55) and (56). In the modified TCP algorithm, at the equilibrium point, equal rates are allocated to sessions with the same end-to-end loss probability. In contrast, in TCP-reno, equal window sizes would be allocated to sessions experiencing identical loss probabilities, should the point of equilibrium be reached.

Another difference between TCP-reno and the MCFC algorithm is in the choice of step size ζ. The value of $\zeta = 0.5$, used in TCP-reno, gives rise to large window size oscillations and prevents convergence to an equilibrium point, while accelerating reaction to changing traffic conditions. In Section 6, using simulation in a simple network, we will show that ζ must be substantially smaller, in order to ensure convergence of the coarse MCFC algorithm. We will also show how multiple values of ζ may be used to combine rapid increase of the window size after a session's initiation, with ultimate convergence. As discussed earlier, the exact MCFC algorithm converges much faster than the coarse algorithm.

Both of the above differences with TCP have a positive impact on the scalability of congestion control algorithms for multicast communications. In a forthcoming paper, we show that multicast communications is more scalable (in terms of the number of receivers) when the applied notion of fairness is based on transmission rates, rather than window sizes, and when the window sizes (or rates) undergo slow adjustments, instead of abrupt changes.

We should also point out that a *packet loss* refers to different events, when considered in the context of TCP and the coarse MCFC algorithm. In TCP, a packet loss is registered whenever a packet is lost or whenever several consecutive acknowledgments are lost, while in the coarse MCFC algorithm a loss exclusively refers to a packet loss in the forward path. It must be clear that congestion on a source-receiver path is best indicated by losses in the forward direction rather than round trip losses, and that the TCP interpretation of packet losses is an implementation necessity rather than a conceptual preference. In this paper, in order to focus on the concepts, we have not addressed the details of protocols needed to implement our algorithms. However, we mention in passing that updating the rate or window size based on forward path loss probability is made possible by executing the update algorithms at the receiver site. This approach has the additional benefit of improving scalability in multicast communications by pushing some of the required processing to the receiver sites.

The above differences notwithstanding, the similarities between the coarse MCFC algorithm and TCP congestion control are significant and allow us to extend our earlier observations to TCP. We have noticed that in the coarse MCFC algorithm, the process of updating rates or window sizes via (40) or (43), is a substitute for estimating the session loss probabilities via (39). In other words, updating a session's window size via (43) in the coarse MCFC algorithm, as well as TCP, amounts to an implicit estimation of the loss probability. In both cases, the end-to-end loss probability summarizes the observations which are used for congestion control.

Controlling congestion based on the end-to-end loss probability has three drawbacks. First, in cases of high packet error rates, e.g. where wireless links are encountered, loss probability is not necessarily indicative of congestion, making congestion control on this basis excessively difficult. Second, the number of packet transmissions required for a reasonable estimation of loss probability is at least an order of magnitude larger than the inverse of the loss probability, itself. This amounts to at least two minutes of observation for estimating a loss probability of 1%, at a session rate of 10 packets/sec. Finally, the loss probability cannot be made too small, for it makes the required observation time even longer. Ironically, in the current Internet, some packet losses are needed in order to practice congestion control and prevent more losses. The only way to avoid this irony and the earlier drawbacks is to provide better end-to-end observations about network congestion status. In the next subsections, other proposals for enhancing congestion observations in the Internet are discussed and compared to the exact MCFC algorithm.

Some of the observations in this section regarding TCP congestion control have been previously discussed in the literature. The impact of making window size increments proportional to τ_s^2 on the relationship between a session's throughput and round-trip time has been studied by Sally Floyd [Flo91], through simulation and analysis of a cascade of congested links. Our results in (55) and (56), which are applicable to an arbitrary topology, essentially agree with those in [Flo91]. Lakshman and Madhow [LM97] provide a detailed analysis and simulation of the performance of TCP congestion control, in which they approximately show that the average throughput of each session s is inversely proportional to τ_s^α, where $1 < \alpha < 2$. In comparison, we have found in (56) that the throughput at the equilibrium point is inversely proportional to τ_s. In regard to TCP performance, the results in [LM97] should be more accurate than our conclusion, for two reasons. First, window sizes in TCP never converge to an equilibrium point whereas (56) specifies the window size (and throughput) at the equilibrium point. Second, in TCP, the link loss probability before and after sessions react to a packet loss is grossly different, due to substantial drop in the volume of traffic. For this reason, there is a considerable time correlation among the dropping of window sizes on sessions sharing a bottleneck link, an issue taken into account in [LM97]. Such correlations are negligible near the equilibrium point of converging algorithms. The equilibrium window size specified in (56) would closely match the average window size of TCP connections, if a step size $\zeta << 0.5$, is used.

5.2. Binary Feedback for Congestion Avoidance

One of the early proposals for explicit notification of congestion is the Binary Feedback Scheme introduced by Ramakrishnan and Jain [RJ88, CJ89]. In this scheme, users are notified about network status through a binary feedback mechanism, i.e., a congestion bit which is set in the packets traversing some congested link. Although our global optimization framework and the MCFC algorithm are different from the methodology and the algorithms in [RJ88], the probing mechanism that we use to collect congestion measures can be viewed as a generalization of the binary feedback. To see the relationship, let the congestion field in probe packets be only one bit. In this case, the incremental congestion cost of the links must also be quantized to two levels; 0 and 1. It follows that once a packet's congestion field is incremented to 1 at some link, it remains 1 regardless of the status of subsequent links, an arrangement identical to the binary feedback in [RJ88].

5.3. Random Early Detection Gateways

The Random Early Detection (RED) scheme [FJ93], proposed by Floyd and Jacobson, enhances the conventional TCP congestion control in two major ways. First, it provides the means of detecting link congestion gradually, by averaging the buffer occupancy over the long run, rather than waiting for buffer overflow and inevitable packet losses to signal congestion abruptly. In this regard, there is a fundamental similarity between the RED scheme and the evaluation of link incremental cost functions in the MCFC algorithm. Second, in the RED scheme, explicit notification of congestion through binary feedback is permitted, thereby detaching congestion notification from packet losses. As discussed in Section 5.1, detaching congestion notification from packet losses resolves several drawbacks inherent to implicit congestion notification via packet losses. Unfortunately, in the proposals [Bra97] that follow the original paper on the RED gateways, this latter aspect of the RED scheme is not promoted.

Although the RED scheme constitutes an important step in the right direction for the improvement of Internet congestion control, we believe that further steps in this direction are needed. Once congestion at the link level is detected gradually and with a quasi-static point of view, it is equally important to convey this gradual change with sufficient granularity to the users, in order to facilitate gradual and smooth reaction to congestion. In other words, a binary feedback stating that the path is either congested or un-congested invites abrupt reaction to such notification and is not commensurate with a smooth and stable control approach.

We believe that timely and efficient congestion control in the Internet would be substantially facilitated by accommodating the use of probe packets, preferably with more than one bit to carry congestion information. Of course, only a small fraction of user packets need to belong to the probing category. In this paper, besides providing a concrete framework for congestion control, we have demonstrated that in order to probe the network congestion status, probing packets need not carry a separate congestion field for each link they traverse. No useful information is missed by providing a single congestion field in the probing packets and adding the incremental congestion cost of the traversed links onto it.

5.4. Dynamic Adaptive Windows

Mitra and Seery, using an elegant analysis of closed queuing networks, have come up with the *Dynamic Adaptive Windows* (DAW), a distributed algorithm for end-to-end calculation of session window sizes [Mit92, MS90, MS93]. Unlike TCP congestion control and the coarse realization of the MCFC algorithm devised in this paper, the DAW algorithm updates the session window sizes based on packet delay measurements, rather than loss observations. In view of the foregoing discussions, the ability to control congestion without relying on packet losses is an attractive feature.

Earlier in Section 4.2, in search of a realization of the MCFC algorithm in the current Internet, we noted that the end-to-end packet delays could be used to detect congestion if the propagation component of the delay was known and if some idea regarding the acceptable range of queuing delays existed. In the DAW scheme, these requirements are satisfied first, by assuming that the round trip propagation time is exactly known for each session and second, by confining the study to small network topologies with specific cross traffic statistics and packet length distributions to enable characterization of queuing delays in a desirable regime of operation, referred to as *moderate usage*. We think that the elaborate study and design in [Mit92, MS90, MS93], when contrasted with its limited application, reinforces the necessity of some form of explicit congestion notification in the increasingly complex Internet.

6. A Brief Simulation Study of Coarse MCFC Algorithm

In this section, we discuss a limited set of simulations for the coarse realization of the MCFC algorithm. The goal of these simulations is to gain some understanding of the behavior of the algorithm, rather than to provide a comprehensive study. In particular, no simulation results are provided for the exact realization of the algorithm.

We consider the window-based implementation of MCFC in (43) with $A_s(w_s)$ and $B_s(w_s)$ given by equations (50) and (51). The round-trip time τ_s is estimated using (4) with a coefficient $\beta = 0.001$. A receiver notifies its source of successful packet delivery by means of an acknowledgment (Ack) packet. We assume that packets are never reordered, and Ack's are never reordered or lost. Hence packet losses are detected via gaps in the sequence number of successive Ack's. We enforce some minimum packet spacing for the window-based implementation of MCFC in order to prevent the phenomenon of packet batching, which has been reported in earlier simulation studies [SZC90].

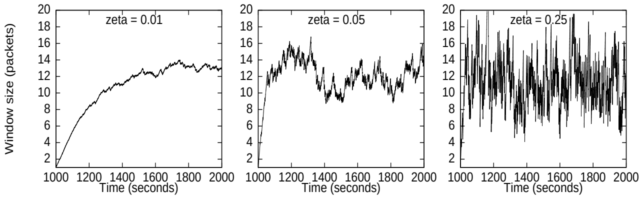

Figure 5. Experiment 1– Evolution of the window of session 1 for different step sizes ζ. Sessions 4–10 are active, but not shown.

We have studied a simple network topology consisting of a single link with a capacity of 1000 packets/sec and a buffer space of 50 packets. Packets are served according to a FIFO scheduling discipline and are dropped from the tail of the link queue in case of overflow.

The single link is shared by 10 sessions, $s = 1, \cdots, 10$. Every session always has data to send after it is activated and its rate is limited by the source congestion algorithm alone. The reverse path delays for Ack's are different for different sources. The fixed and random components of these delays are chosen such that the round-trip times of the sources are as follows : $\tau_s \approx 200$ msec for $s = 2$ and $s = 6, \cdots, 10$, $\tau_s \approx 100$ msec for $s = 1$, $\tau_s = 300$ msec for $s = 3$, $\tau_s = 400$ msec for $s = 4$, and $\tau_s = 500$ msec for $s = 5$.

In the first experiment, sessions 4–10 are started and the network is allowed to reach a stable operating point, then session 1 is activated at time $t = 1000$ sec. Using $\eta = 50$ and $\zeta = 0.01,\ 0.05$, and 0.25, we observe the effect of the step size ζ in (50) and (51) on the stability and speed of convergence of the algorithm. We observe from Figure 5 that as ζ increases, session 1's window reaches its steady state value faster but the size of oscillations in the steady state increases.

In the second experiment, we have tried to combine the benefits of a large ζ (fast rise to steady state) and a small ζ (small oscillations). Hence we have used $\zeta = 0.25$ when a session is first activated and have switched to $\zeta = 0.01$ at a later stage. The criterion that we have applied for this switching to takes place is the number of losses experienced by a session. A threshold of 12 losses has been used in this simulation. The threshold value has to be chosen in a way such that the session's window reaches a given neighborhood of the steady state value before the switching takes place. It can be shown that, with $A_s(w_s)$ and $B_s(w_s)$ chosen as in (50) and (51), the threshold, on the average, depends only on the initial value of ζ and is independent of the final value of the window, the link capacity, or the buffer size.

For Figures 6.A and 6.B, sessions 4 through 10 have reached their steady states at $t = 1000$ sec. Sessions 1, 2, and 3 are activated at $t = 2000$ sec, $t = 3000$ sec, and $t = 4000$ sec, respectively. In order to study the algorithm behavior under severe congestion, an uncontrolled source with a rate of 500 packets/sec is activated at time $t = 4000$

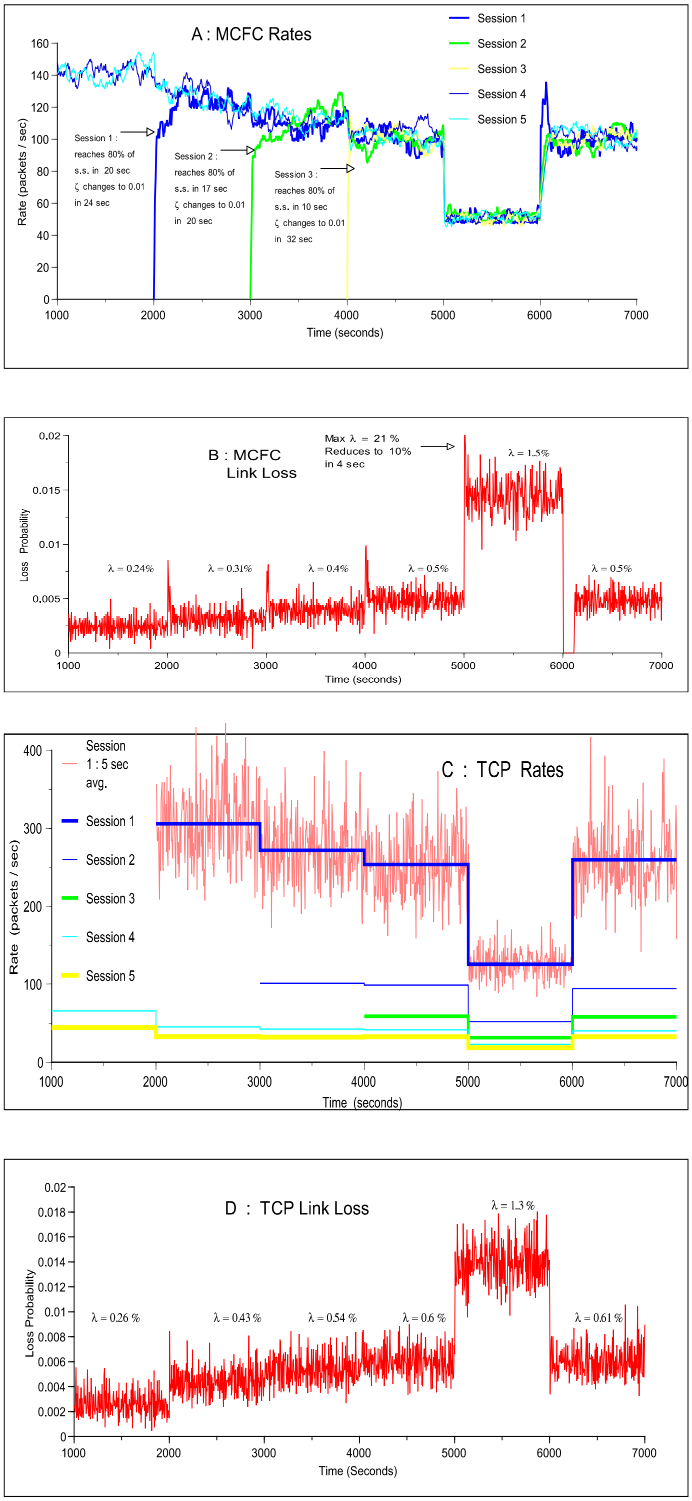

Figure 6. Experiment 2 : A. Rates of sessions $1 - 5$ (averaged over 5 second intervals) for MCFC. B. The loss probability of the link (averaged over 5 second intervals) for MCFC. C. Rates of sessions $1 - 5$, averaged over 1000 second intervals, for TCP-reno. The rate of session 1 (averaged over 5 second intervals) is also shown. D. The loss probability of the link (averaged over 5 second intervals) for TCP-reno.

sec, and stopped at $t = 5000$ sec. The rate of sessions $1 - 5$, averaged over 5 second intervals is illustrated in Figure 6.A. We observe that at every point of time, all the active sessions attain the same rate, in spite of differences in their round-trip times. We also observe that every incoming session is allowed its fair share of the link bandwidth and it attains this share quite rapidly. Moreover, all sessions react rapidly and uniformly to changes in the quasi-static state of the network, as exemplified by the activation of sessions 1, 2, and 3, and by the activation and the termination of the uncontrolled source. Figure 6.B illustrates the loss probability at the link. The most significant observation in this figure is the sudden jump in the loss probability when the uncontrolled source is activated. However, small loss probability at the queue is restored within a few seconds.

In Figures 6.C and 6.D, we present the results for the same network activity and configuration, but with sessions running the TCP-reno algorithm, modified to fit our simulation model. Here we observe that link bandwidth is not shared fairly among the sessions – sessions with larger round-trip times attain lower average rates. The behavior of the link loss probability in Figure 6.D is similar to that in 6.B, though we do not observe the sharp jump at $t = 5000$ sec, indicating that TCP reacts faster to the build up of congestion.

In conclusion, we add that the speed of the coarse MCFC algorithm may be further improved by incorporating the TCP slow start phase into it, or by using a function $a_s(r_s) = \zeta{\cdot}\eta/r_s^{\alpha}$, with some $\alpha < 1$, instead of the $\alpha = 1$ used in (47). We assert once again that the convergence and the reaction speed of the exact MCFC algorithm is inherently faster than the coarse algorithm studied in this simulation.

7. Conclusion

We have developed a class of optimal algorithms for end-to-end congestion control at the transport layer of IP networks. The global optimization framework used for this purpose, allowed us to systematically address issues of fairness and user priority. Although the proposed algorithms do not require non-FIFO switches, we have shown that they can provide fair services to the users or help enforce certain priority options among them. These algorithms are realizable in both a coarse and an exact fashion, using implicit or explicit congestion information. Therefore, they facilitate an objective evaluation of the performance improvement that explicit congestion notification can bring to the Internet.

As a significant result, we noticed that TCP-reno algorithm, once modified to make its session throughput independent of round trip times, belongs to the class of MCFC algorithms. We provided a mathematical characterization of session throughputs or window sizes in terms of loss probabilities, for both TCP and its modified version. Although these results are not precise because of the large step sizes in TCP-reno algorithm, they are applicable to arbitrary network topologies.

In a forthcoming paper, we use the methodology developed in this paper to study congestion control for multicast communications in the Internet and the associated problems of fairness and scalability.

References

[BG92] D. P. Bertsekas and R. G. Gallager. *Data Networks*. Prentice-Hall, Englewood Cliffs, N.J., 1992.

[Bra97] B. Braden, et al. Recommendations on queue management and congestion avoidance in the Internet. Internet Draft draft-irtf-e2e-queue-mgt-00.ps, March 1997. Expires on September 25, 1997.

[BT89] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[CJ89] D. M. Chiu and R. Jain. Analysis of the increase and decrease algorihtms for congestion avoidance in computer networks. *Compter Networks and ISDN systems*, 17:1–14, 1989.

[FF96] K. Fall and S. Floyd. Simulation-based comparison of tahoe, reno and sack TCP. *ACM Computer Communications Review*, 26(3):5–21, July 1996.

[FJ93] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[Flo91] S. Floyd. Connections with multiple congested gateways in packet-switched networks part1: One-way traffic. *ACM Computer Communication Review*, 22(5):30–47, October 1991.

[Flo94] S. Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, October 1994.

[Gal77] R. G. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Trans. Comm.*, 23:73–85, 1977.

[GG80] R. G. Gallager and S. J. Golestani. Flow control and routing algorithms for data networks. In *Proc. 5'th Internat Conf. Comput. Comm.*, pages 779–784, 1980.

[Gol79] S. J. Golestaani. *A Unified Theory of Flow Control and Routing in Data Communication Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, December 1979.

[Gol94] S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *IEEE INFOCOM'94*, pages 636–646, 1994.

[Jac88] V. Jacobson. Congestion avoidance and control. In *Proc. ACM SIGCOMM'88 Conf.*, pages 158–173, 1988.

[Kes91] S. Keshav. A control theoretic approach to flow control. In *Proc. ACM SIGCOMM'91 Conf.*, pages 3–15, September 1991.

[LM97] T. V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEE/ACM Transactions on Networking*, 5(3):336–350, June 1997.

[Lue89] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, 1989.

[Mit92] D. Mitra. Asymptotically optimal design of congestion control for high speed data networks. *IEEE Transactions on Communications*, 40(2):301–311, February 1992.

[MS90] D. Mitra and J. B. Seery. Dynamic adaptive windows for high speed data networks. In *Proc. ACM SIGCOMM'90 Conf.*, pages 30–40, 1990.

[MS93] D. Mitra and J. B. Seery. Dynamic adaptive windows for high speed data networks with multiple paths and propagation delays. *Computer Networks and ISDN Systems*, (25):663–679, 1993.

[RJ88] K. K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. In *Proc. ACM SIGCOMM'88 Conf.*, pages 303–313, 1988.

[SZC90] S. Shenker, L. Zhang, and D.D. Clark. Some ovservations on the dynamics of a congestion control algorithm. *ACM Computer Communication Review*, pages 30–39, October 1990.

[ZDE$^+$93] L. Zhang, S. Deering, D. Estring, S. Shenker, and D. Zappala. Rsvp: A new resource reservation protocol. *IEEE Networks*, 7(5), September 1993.

[Zha91] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching. *ACM Transactions on Computer Systems*, 9(2):101–124, May 1991.