Increase-Decrease Congestion Control for Real-time Streaming: Scalability

Dmitri Loguinov
Computer Science Department
City University of New York
New York, NY 10016
csdsl@cs.ccny.cuny.edu

Hayder Radha
Dept. of Electrical & Computer Engineering
Michigan State University
East Lansing, MI 48824
radha@egr.msu.edu

Abstract – **Typically, NACK-based congestion control is dismissed as being not viable due to the common notion that "open-loop" congestion control is simply "difficult." Emerging real-time streaming applications, however, often rely on rate-based flow control and would benefit greatly from scalable NACK-based congestion control. This paper sheds new light on the performance of NACK-based congestion control and measures the amount of "difficulty" inherently present in such protocols. We specifically focus on *increase-decrease* (I-D) congestion control methods for real-time, rate-based streaming. First, we introduce and study several new performance measures that can be used to analyze the class of general I-D congestion control methods. These measures include *monotonicity* of convergence to fairness and *packet-loss scalability* (explained later in the paper). Second, under the assumptions that the only feedback from the network is packet loss, we show that AIMD is the only TCP-friendly method with monotonic convergence to fairness. Furthermore, we find that AIMD possesses the best packet-loss scalability among all TCP-friendly binomial schemes [2] and show how poorly all of the existing methods scale as the number of flows is increased. Third, we show that if the flows can obtain the knowledge of an additional network parameter (i.e., the bottleneck bandwidth), the scalability of AIMD can be substantially improved. We conclude the paper by studying the performance of a new scheme, called *Ideally-Scalable Congestion Control* (ISCC), both in simulation and a NACK-based MPEG-4 streaming application over a Cisco testbed.**

I. Introduction

Congestion is an inherent property of the currently best-effort Internet. Consequently, transport protocols (such as TCP) commonly implement *congestion control*, which refers to end-to-end algorithms executed by a protocol in order to properly adapt the sending rate of a network flow to the available bandwidth in the path along which the flow sends its packets. Protocols with ACK-based flow control utilize one or another version of TCP-friendly congestion control, which includes Jacobson's modifications to TCP [1], [9], TCP-like congestion control (e.g., [19]), increase-decrease algorithms (e.g., [2], [3], [5], [8], [10], [12], [14], [21]), and equation-based methods (e.g., [7], [15]). These algorithms are shown to work well in the environment where the sender relies on "*self-clocking*," which refers to the use of positive acknowledgements in congestion control.

However, current real-time streaming applications in the Internet [18] typically rely on NACK-based (i.e., rate-based) flow control[1], for which congestion control either does not exist, or assumes a very rudimentary form [18]. Furthermore, congestion control in NACK-based applications is typically labeled as being "difficult" due to the "open-loop" operation of its flow control, and the actual extent of "difficulty" remains neither documented nor measured.

At the same time, before emerging real-time streaming applications can gain wide-spread acceptance, we believe that they first must implement some form of scalable congestion control. Therefore, in this paper, we undertake an analysis and performance study that sheds the light on both the exact difficulties found in "open-loop" congestion control and the extent of penalty incurred by a NACK-based protocol in an Internet-like environment. In the course of our investigation, we found that traditional NACK-based congestion control possessed poor scalability (i.e., their use resulted in high packet loss when the number of simultaneous flows was large) and that the stability of existing NACK-based schemes was much lower than that of similar ACK-based schemes. Note that this paper does not study a fundamental question of whether NACK-based congestion control can achieve the same level of stability as its ACK-based counterparts, but rather investigates previously-undocumented drawbacks of NACK-based congestion control and attempts to improve the performance of the existing schemes in rate-based applications.

Studying new congestion control methods in this paper, we sometimes drift away from TCP-friendly schemes. Hence, we must mention a few words about why find such practice acceptable. We argue that in the future Internet, it is quite possible that UDP traffic will *not* compete with TCP in the same router queues (e.g., DiffServ may be used to separate these types of traffic at the router level). This intuition is driven by the fact that real-time flows have substantially different delay requirements from those of TCP, and it may not be practical to mix the two types of traffic in the same queues. Furthermore, NACK-based applications are unlikely to be fully TCP-friendly, because they often do not follow TCP's *fast retransmit* and *timeout backoff* algorithms and do not rely on the "packet-conservation" principle [9] in their flow control.

The remainder of the paper is organized as follows. Section II provides the necessary background on increase-decrease (I-D) congestion control. In section III, we define the notion of *monotonic convergence* to fairness of general I-D congestion control and derive certain desired properties of control functions that guarantee such monotonic convergence. We next focus on binomial algorithms [2] in section IV and, under simple assumptions, derive their average link utilization and packet loss rate in the stable state. In section V, we study packet-loss scalability of binomial congestion control and show that AIMD possesses the best scalability among all TCP-friendly schemes. In addition, we show that to achieve optimal scalability (i.e., constant packet loss), a congestion control scheme must have the knowledge of the bottleneck bandwidth.

[1] Note that ACK-based flow control could be used in real-time streaming, but it typically results in some form of QoS penalty (such as longer startup delays, more frequent buffer underflow events, etc.).

 0-7803-7476-2/02/$17.00 © 2002 IEEE.

In section VI, we investigate the feasibility of using real-time bottleneck bandwidth estimates as a supplement to binomial congestion control and study whether the new schemes can achieve better scalability than AIMD in a real network. We conclude the paper in section VII.

II. Background

Within the class of end-to-end congestion control protocols, we specifically focus on the class of *increase-decrease* (I-D) methods. I-D congestion control implements a simple reactive control system, which responds to congestion by decreasing the sending rate and responds to the absence of congestion by increasing the sending rate. Hence, at any stage, the decision of I-D congestion control is binary.

Furthermore, the increase and decrease functions are *local* [2], [5], which means that they only use the local state of a flow in computing the next value of the sending rate. In addition, I-D congestion control usually assumes a *memoryless* model [2], [5], in which the amount of increase and decrease is based only on the value of the current sending rate rather than on the history of the sending rate (e.g., several flavors of "AIMD with history" are examined in [11], [12]). In this paper, we explicitly assume a local and memoryless model of I-D congestion control.

To prevent high-frequency oscillations on timescales smaller than it is needed to receive the feedback from the network, I-D congestion control is executed on discrete timescales of R time units long. Typically, R is a multiple of the round-trip delay (RTT) and in many cases, simply equals the RTT.

Many papers study congestion control in the context of *window-based* flow control [2], [8], [12], [21] and apply I-D formulas to the size of congestion window *cwnd*. In such notation, assuming that the size of congestion window *cwnd* during interval i for a particular flow is given by w_i, I-D congestion control can be summarized as:

$$w_{i+1} = \begin{cases} w_i + W_I(w_i), & f = 0 \\ w_i - W_D(w_i), & f > 0 \end{cases}. \qquad (1)$$

where f is the congestion feedback (positive values indicate congestion), and W_I and W_D are the increase and decrease functions of *window-based* I-D congestion control, respectively. In practice, feedback f is usually equal to the packet loss rate observed by the flow during the last interval (i.e., interval i).

Since our work focuses on *rate-based* streaming applications (in which *cwnd* has little meaning), we must write an equivalent formulation of increase-decrease congestion control using the value of each flow's sending rate r_i instead of congestion window w_i. The conversion from the packet-based notation to the rate-based notation is straightforward, i.e., each unit of w_i is equivalent to a rate of MTU/RTT bits/s, where the MTU (Maximum Transmission Unit) is given in bits and the RTT is given in seconds. In other words, $r_i = MTU/RTT w_i$.

Therefore, assuming that r_i is the sending rate of a particular flow during discrete interval i, the I-D congestion control (1) for that flow can be re-written as:

$$r_{i+1} = \begin{cases} r_i + R_I(r_i), & f = 0 \\ r_i - R_D(r_i), & f > 0 \end{cases}, \qquad (2)$$

where R_I and R_D are the increase and decrease functions of *rate-based* I-D congestion control, respectively.

One special case of I-D congestion control is given by *binomial algorithms*, where the increase and decrease functions are simple power functions [2]:

$$\begin{cases} W_I(w) = \alpha w^{-k} \\ W_D(w) = \beta w^{l} \end{cases} \text{ or } \begin{cases} R_I(r) = \lambda r^{-k} \\ R_D(r) = \sigma r^{l} \end{cases}, \qquad (3)$$

where all constants α, β, λ, σ are positive. For binomial algorithms, the difference between the two notations lies only in the constants in front of the corresponding power functions. Hence, the conversion from the window-based to the rate-based notation is supplied by the following formulas:

$$\lambda = \alpha\left(\frac{MTU}{RTT}\right)^{k+1} \text{ and } \sigma = \beta\left(\frac{MTU}{RTT}\right)^{1-l}. \qquad (4)$$

Throughout the rest of the paper, we will use both versions of binomial algorithms in (3), sometimes referring to constants (λ,σ) instead of constants (α,β), while keeping in mind the conversion in (4).

A special case of binomial congestion control that is implemented in TCP is called AIMD (Additive Increase, Multiplicative Decrease) [5], [9]. In AIMD, k equals 0, i.e., $W_I(w) = \alpha$ ($\alpha > 0$), and l equals 1, i.e., $W_D(w) = \beta w$ ($0 < \beta < 1$). AIMD(α,β) is *TCP long-term fair*[2], if it achieves the same average throughput when competing with a TCP connection under the same end-to-end conditions. The necessary condition for such long-term fairness is [8], [12], [21][3] $\alpha = 3\beta/(2-\beta)$. On the other hand, for binomial congestion control (3) to be TCP-friendly, Bansal *et al*. [2] show that $k + l$ must be equal to 1. Among such (non-AIMD) TCP-friendly binomial congestion control, they propose two methods called IIAD (Inverse Increase, Additive Decrease) with $k = 1$, $l = 0$, and SQRT (Square Root) with $k = l = ½$.

Finally, we should mention that the analysis of increase-decrease congestion control typically assumes an ideal network with *synchronized* and *immediate* feedback [2], [5], [10], [11], [12]. *Synchronized* feedback means that all flows sharing a congested link receive notifications about packet loss at the same time. *Immediate* feedback means that if the capacity of any link along an end-to-end path is exceeded during interval i, feedback f is positive for interval i. Under these ideal conditions, Chiu and Jain [5] show that all AIMD schemes converge to a fair state. In addition, Bansal *et al*. [2] show that for binomial algorithms (3) to converge to fairness, $k + l$ must be strictly greater than zero.

III. General I-D Control

Not all increase-decrease functions R_I and R_D guarantee convergence to *fairness*. In the context of I-D congestion control, convergence to fairness is usually defined as the ability of any number of identical flows sharing a common bottleneck

[2] Sometimes called *TCP-compatible* [2], [8] or *TCP-friendly* [21].

[3] Note that some papers [2], [20], [21] use a different notation, in which $W_D(w) = (1-\beta)w$ and this formula has a different form. Furthermore, if the rate of AIMD is dominated by timeouts, the formula assumes yet another form [21].

link to reach a state in which their rates become equal and stay equal infinitely long. Even though in practice this is a very difficult goal to achieve, under the ideal conditions of synchronized and immediate feedback, many schemes can guarantee convergence to fairness.

One of the interesting properties of I-D congestion control that we introduce in this paper is the ability of a scheme to approach fairness *monotonically*, i.e., if the fairness during interval i is given by f_i, $0 \le f_i \le 1$, then the following conditions are necessary for monotonic convergence:

$$\forall i : f_{i+1} \ge f_i \text{ and } \lim_{i \to \infty} f_i = 1 . \quad (5)$$

Generally, monotonic convergence is not necessary, but it is beneficial, because non-monotonic convergence tends to temporarily drive the system into extremely unfair states (i.e., one flow receiving much higher bandwidth), especially in the presence of random packet losses and heterogeneous feedback delays. Later in this paper, we will relax the above condition of monotonic convergence, but will keep the rest of the results in this section as they are applicable to both binomial algorithms and the ideally-scalable schemes studied in section V.

It is common [2], [12] to examine the case of two flows sharing a link, since the extension to n flows can be easily performed by considering flows pair-wise. It is also common to use a continuous fluid approximation model [2] and disregard the discrete nature of packets (i.e., all packets are infinitely divisible). Furthermore, in this paper, we use a max-min fairness function f_i instead of Chiu's fairness index [5]. Recall that max-min fairness of n flows with *non-zero* sending rates $(x_1,\ldots, x_n)$ is given by:

$$f = \min_{i \ne j}\left(x_i / x_j\right). \quad (6)$$

Consider two flows X and Y sharing a bottleneck link under the above assumptions. Suppose that during interval i, the flows' sending rates are given by x_i and y_i, respectively. To help us understand the behavior of a two-flow I-D control system, we use Figure 1 from [5]. In the figure, the axes represent the sending rate of each of the two flows. Furthermore, line $y = x$ is known as the *fairness line* and represents points (x, y) in which fairness f equals 1. Assuming that the capacity of the bottleneck link is C, line $x + y = C$ is called the *efficiency line* and represents points in which the bottleneck link is about to overflow. Given a particular point $P_i = (x_i, y_i)$ in the figure, line $y = mx$ connecting P_i to the origin is called the *equi-fairness line* (i.e., points along the line have the same fairness $f_i = x_i/y_i = 1/m$). Furthermore, we define *efficiency* e_i of point P_i as the combined rate of both flows in that point, i.e., $e_i = x_i + y_i$.

A. Decrease Function

To ensure monotonic convergence and proper response to congestion signals, the following four conditions must hold during each *decrease* step assuming that the system is in some point P_i just before the decrease step.

First, the efficiency in the new state must be strictly less than that in the old state, i.e., $e_{i+1} < e_i$. This condition ensures that flows backoff during congestion. Second, the fairness must not decrease in the new state, i.e., $f_{i+1} \ge f_i$. This condition guarantees monotonic convergence to fairness, and as pointed

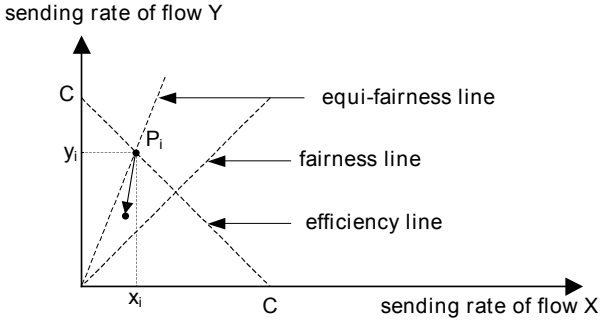

Figure 1. Two-flow I-D control system.

out before, although desired, it is often not available in practice. Consequently, we will relax this condition later in the paper. Third, to properly maintain convergence, the system must not arbitrarily cross or oscillate around the fairness line, i.e., it must stay on the same side of the fairness line at all times. For the case in Figure 1, we can write: $(y_i > x_i) \Rightarrow (y_{i+1} > x_{i+1})$. Finally, the system must not allow rates below or equal to zero, i.e., given an arbitrary state with $x_i > 0$ and $y_i > 0$, we must guarantee that $y_{i+1} > 0$ and $x_{i+1} > 0$.

The first condition is equivalent to:

$$x_{i+1} - x_i + y_{i+1} - y_i = -R_D(x_i) - R_D(y_i) < 0 , \quad (7)$$

which can be satisfied with any positive function $R_D(x) > 0$, $\forall x > 0$. The second condition is equivalent to:

$$\frac{x_{i+1}}{y_{i+1}} \ge \frac{x_i}{y_i}, x_i > 0, y_i > 0. \quad (8)$$

Expanding the last inequality using (2) and generalizing by dropping the indexes (the inequality depends only on x_i and y_i), we get:

$$xR_D(y) - yR_D(x) \ge 0 , \text{ for all } x > 0, y > 0, x < y. \quad (9)$$

Writing $y = x + \Delta x$, for $\Delta x > 0$:

$$x\left(R_D(x + \Delta x) - R_D(x)\right) - \Delta x R_D(x) \ge 0 , \quad (10)$$

$$\frac{R_D(x + \Delta x) - R_D(x)}{\Delta x} \ge \frac{R_D(x)}{x} , \text{ for } x > 0, \Delta x > 0. \quad (11)$$

Restricting $R_D(x)$ to be a differentiable function for all $x > 0$, (11) is equivalent to:

$$R'_D(x) \ge \frac{R_D(x)}{x} , \text{ for all } x > 0. \quad (12)$$

Bringing $R_D(x)$ to the left and taking the integral (both x and $R_D(x)$ are known to be positive):

$$\int \frac{dR_D(x)}{R_D(x)} \ge \int \frac{dx}{x} , \text{ for all } x > 0. \quad (13)$$

$$\ln R_D(x) \ge \ln x + m_1 , \text{ for all } x > 0. \quad (14)$$

$$R_D(x) \ge m_2 x , \text{ for all } x > 0. \quad (15)$$

The result in (15) shows that the original condition (12) restricts $R_D(x)$ to grow no slower than some linear function m_2x.

Using similar derivations, we find that the third condition (i.e., the non-cross-over condition) results in:

$$R'_D(x) < 1 , \text{ for all } x > 0, \quad (16)$$

0-7803-7476-2/02/$17.00 © 2002 IEEE.

which means that $R_D(x)$ must grow slower than function x (i.e., the slope of $R_D(x)$ in all points $x > 0$ must be less than 1). Finally, the fourth condition

$$x - R_D(x) > 0 \text{, for all } x > 0 \tag{17}$$

is automatically satisfied by combining (12) and (16) above.

To summarize by combining (15) and (16), function $R_D(x)$ must be positive and differentiable for all values of $x > 0$, and must be an asymptotically (i.e., for substantially large x) linear function of x, with the slope strictly less than 1. For example, AIMD function $R_D(x) = \sigma x$ clearly satisfies these conditions for $0 < \sigma < 1$.

B. Increase Function

The analysis of increase function $R_I(x)$ is similar to the above. This time, instead of four conditions, we have only three. First, the efficiency in the new state must increase (i.e., $e_{i+1} > e_i$), which guarantees that flows will probe for new bandwidth in the absence of congestion. Second, the fairness must not decrease (i.e., $f_{i+1} \geq f_i$), which is the result of the same monotonicity requirement as before. And third, the system must not cross the fairness line (i.e., $y_{i+1} > x_{i+1}$). Crossing the fairness line violates monotonic converge to fairness and, as we will see later, never happens in practice (i.e., among binomial schemes).

The first condition is satisfied with any positive function $R_I(x)$, i.e., $R_I(x) > 0$, $\forall x > 0$. The second condition is the opposite of (12) due to a different sign in (2):

$$R_I'(x) \leq \frac{R_I(x)}{x} \text{, for all } x > 0. \tag{18}$$

Finally, the third condition is similar to (16), but assumes the following shape:

$$R_I'(x) > -1 \text{, for all } x > 0. \tag{19}$$

Using (18), we find that R_I must grow no faster than some linear function $m_3 x$ and using (19), R_I cannot decay quicker than $-x$. For example, AIMD increase function $R_I(x) = \lambda$ again satisfies all conditions of monotonic convergence for $\lambda > 0$. We will look at other examples in the next section while studying binomial congestion control methods [2].

C. Convergence

Note that the above conditions still do not guarantee convergence to fairness. In other words, the conditions guarantee that if the system converges, it will do so monotonically, but the fact of convergence has not been established yet. Hence, we impose a final restriction on R_D and R_I – either the decrease or the increase step must strictly improve fairness, i.e., one of (12), (18) must be a *strict* inequality. If (12) is made into a strict inequality, we can no longer satisfy the condition in (16). Consequently, (12) must remain in its present form, and (18) must become a strict inequality.

IV. Properties of Binomial Algorithms

A. Overview

Consider binomial algorithms in (3). Clearly, both functions R_I and R_D are positive for $x > 0$ and therefore, satisfy the first condition. The second condition (i.e., monotonically non-decreasing fairness) results in the following restrictions on k and l from applying (12) and the strict form of (18):

$$\begin{cases} -\lambda k x^{-(k+1)} < \lambda x^{-k} / x \\ \sigma l x^{l-1} \geq \sigma x^{l} / x \end{cases} \Rightarrow \begin{cases} k > -1 \\ l \geq 1 \end{cases}. \tag{20}$$

The third (i.e., non-cross-over) condition derived from (16) and (19) restricts l even further, but does not impose any limit on k (assuming sufficiently large x):

$$\begin{cases} k\lambda < x^{k+1} \\ l\sigma < x^{1-l} \end{cases} \Rightarrow \begin{cases} k = anything \\ l \leq 1 \end{cases}. \tag{21}$$

Note that restriction on l in (21) is dictated by the fact that sending rate x of a flow is not limited *a-priori* and the selection of a *positive* constant σ such that it is less than x^{1-l}/l, for substantially large $x > 0$, is feasible only when power $1-l$ is strictly non-negative.[4] Later in this paper, we will show how restriction $l \leq 1$ can be lifted and what kind of advantages such schemes bring to congestion control protocols.

Consequently, assuming that the upper limit on x is not known, for a binomial algorithm to possess monotonic convergence to fairness, both (20) and (21) must be satisfied. In practice, this means that l must be strictly 1. Knowing that for TCP-friendly binomial congestion control $k + l$ must be one [2], we arrive at the fact that AIMD is the *only* TCP-friendly binomial algorithm with monotonic convergence to fairness. Hence, for the rest of the paper, we will study schemes with non-monotonic convergence to fairness, because we want to go beyond what AIMD has to offer.

In the absence of monotonic convergence, [2] shows that the necessary condition for convergence is $k + l > 0$ (i.e., flows make due progress towards the fairness line not necessarily at *every* step, but between every two consecutive *decrease* steps). Hence, dropping the monotonicity requirement and combining (21) with the convergence rule $k + l > 0$, we notice that the necessary restrictions on k and l for convergence of *non-monotonic* binomial algorithms are: $k > -1$ and $l \leq 1$.

B. Efficiency

The average efficiency is an important property of a congestion control scheme, which reflects how well the scheme utilizes the bottleneck bandwidth in the stable state. Clearly, higher efficiency is more desirable (but not necessarily at the expense of other properties of the scheme, such as packet loss or convergence speed). Formulas derived in this section not only help us study the efficiency of binomial schemes, but also are a necessary background for our packet-loss scalability analysis in the next section.

We define the *average efficiency* of a scheme as the percentage of the bottleneck link utilized by the scheme over a long period of time once the scheme has reached its stable state. In the stable state, each flow's sending rate will oscillate between two points, which we call the *upper point* (U) and the *lower*

[4] Note that we implicitly assume that x is limited from below by some constant x_{min}. In window-based congestion control, x_{min} is equivalent to one unit of *cwnd* (i.e., *MTU/RTT*), and in rate-based congestion control, x_{min} is the minimum rate at which real-time material can be received (e.g., the rate of the base video layer).

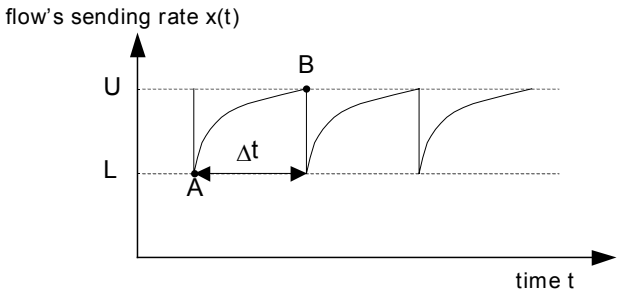

Figure 2. Oscillation of the sending rate in the stable state.

point (*L*) as shown in Figure 2. When a single flow is present in the network, *U* equals the capacity of the bottleneck link *C*. When *n* flows compete over a shared link of capacity *C*, *U* equals *C*/*n* for each flow (because the flows have reached fairness by this time). In both cases, $L = U - \sigma U^l$ according to (3). In addition, since the pattern in Figure 2 is repetitive, it is sufficient to determine the average throughput of a flow during a single oscillation (i.e., between points *A* and *B*) rather than over a longer period of time. Note that in the window-based notation of congestion control, the maximum capacity of the link is given by $W = C \cdot RTT/MTU$.

Using a continuous fluid approximation and results from [2], each flow's rate $x(t)$ *during the increase phase* (i.e., between points *A* and *B*) is given by:

$$x(t) = \left(\frac{\lambda(k+1)t}{R}\right)^{\frac{1}{k+1}}, \tag{22}$$

where *R* is a fixed duration of the control interval (which is typically equal to the value of the RTT). Following [2], the duration between points *A* and *B* in Figure 2 is:

$$\Delta t = \frac{U^{k+1}\left(1-\left(1-\sigma U^{l-1}\right)^{k+1}\right)R}{\lambda(k+1)}, \tag{23}$$

and the total amount of bits transmitted during the same interval is:

$$X = \frac{U^{k+2}\left(1-\left(1-\sigma U^{l-1}\right)^{k+2}\right)R}{\lambda(k+2)}. \tag{24}$$

Consequently, we derive that the flow's average sending rate during the interval is $X/\Delta t$ and the *average efficiency* (i.e., percent utilization) of a binomial congestion control scheme is:

$$e = \frac{X}{U\Delta t} = \frac{(k+1)\left(1-\left(1-\sigma U^{l-1}\right)^{k+2}\right)}{(k+2)\left(1-\left(1-\sigma U^{l-1}\right)^{k+1}\right)}. \tag{25}$$

Note that (25) can be converted to the window-based notation by replacing σ with β and rate *U* with its window equivalent. We also note that for large *n*, the exact model of efficiency *e* in (25) becomes inapplicable when $U = C/n$ drops below $\sigma^{1/(1-l)}$. We can no longer use any of the above derivations due to the fact that $1-\sigma(C/n)^{l-1}$ becomes negative, which is caused by the "drop-below-zero" effect (i.e., rate $x(t)$ becomes negative) that we tried to avoid before in (17). This condition was automatically satisfied given monotonic convergence to fairness in (12), but in the absence of monotonicity, we must explicitly restrict *n* to the following:

$$n < \frac{C}{\sigma^{1/(1-l)}} = \frac{W}{\beta^{1/(1-l)}}. \tag{26}$$

We next focus on simplifying the expression in (25). Equation (25) contains two terms of the form $1-(1-z)^q$, which can be expanded using Taylor series to:

$$1-(1-z)^q = qz\left(1-\frac{q-1}{2}z+\frac{(q-1)(q-2)}{6}z^2\ldots\right). \tag{27}$$

Note that for $l < 1$, the value of *z* is less than 1, which means that the higher order terms in (27) get progressively smaller. Hence, by keeping the first two terms[5] in (27), we arrive at the following approximation to the exact formula in (25):

$$e = 1-\frac{\sigma U^{l-1}}{2-k\sigma U^{l-1}}. \tag{28}$$

To perform a self-check, we plug AIMD parameters ($l = 1$, $k = 0$) into (28) and get the familiar (and exact) formula of the average efficiency of an AIMD scheme: $e = (2-\beta)/2$ (recall that $\sigma = \beta$ in AIMD).

C. Packet Loss

The amount of packet loss during the stable state is another important property of a congestion control scheme. Consider one oscillation cycle between points *A* and *B* in Figure 2 and the case of a single flow. The maximum amount of overshoot under non-ideal (i.e., non-continuous) conditions will be the value of the increase function just before the flow reaches its upper boundary *C* in point *B*. Hence, the amount of the maximum overshoot for a single flow is given by $\lambda C^{-k}R$, where *R* is the fixed duration between control actions. Knowing how many bits *X* were sent by the flow during the same interval of duration Δt, we can write the average percentage of lost data p_1 using (24) and assuming the worst case of the maximum overshoot as:

$$\begin{aligned} p_1 &= \frac{\lambda C^{-k}R}{X+\lambda C^{-k}R} \\ &= \frac{\lambda^2(k+2)}{C^{2k+2}\left(1-\left(1-\sigma C^{l-1}\right)^{k+2}\right)+\lambda^2(k+2)} \\ &\approx \frac{\lambda^2(k+2)}{C^{2k+2}\left(1-\left(1-\sigma C^{l-1}\right)^{k+2}\right)}, \end{aligned} \tag{29}$$

when $\lambda C^{-k}R << X$. In particular, for AIMD schemes, the packet loss rate in the worst case is given by:

$$p_1 = \frac{2\lambda^2}{C^2\sigma(1-\sigma)+2\lambda^2} \approx \frac{2\lambda^2}{C^2\sigma(1-\sigma)} = \frac{2\alpha^2}{W^2\beta(1-\beta)}. \tag{30}$$

A close look at the last equation reveals that as the number of flows increases (i.e., *C* is replaced by *C*/*n*), AIMD's packet loss rate will also increase. Furthermore, the amount of in-

[5] A one-term approximation used in [2] typically possesses an insufficient accuracy.

crease is proportional to n^2, where n is the number of flows. This confirms a well-known fact that AIMD scales as n^2 when it comes to packet loss [14]. Note that as $n \to \infty$, the amount of overshoot $\lambda C^{-k} R$ will become large compared to the value of X, and the approximations above will no longer work. However, the exact formulas in (29) and (30) will asymptotically approach the correct value of 100%.

Consider a simple explanation of why AIMD scales quadratically. In AIMD, the increase in packet loss by a factor of n^2 comes from two places – from the reduction in the number of discrete increase steps N during interval Δt by a factor of n (because the increase distance $U–L$ becomes n times smaller), and from the reduction of duration Δt by the same factor of n (due to the same reason). As a result, the number of bits sent during the interval (which is proportional to $N\Delta t$) is reduced by a factor of n^2, and the amount of overshoot is unchanged (i.e., λR). Consequently, the total amount of lost packets relative to the number of sent packets is increased by a factor of n^2.

There are two reasons why we do not see this kind of performance degradation in practice. First, our results in (30) are based on a continuous fluid model, which assumes that packets are infinitely divisible. However, in practice, this approximation is true only when the amount of increase λR is negligible compared to the difference between the upper and lower limits, i.e., $U–L$ in Figure 2. Hence, when the number of *discrete* increase steps N becomes equal to 1 (or approaches 1), it can no longer be reduced by a factor of n, because it must remain an integer. Taking into account a fixed value of $N = 1$, the increase in packet loss becomes a linear rather than a quadratic function of n.

Second, most protocols employing AIMD rely on positive ACKs in implementing congestion control. This "self-clocking" [9], or "packet conservation," is capable of significantly improving the scalability aspects of AIMD, because the sender does not inject more packets into the network than the network can handle at any given time. "Open-loop" congestion control (i.e., NACK-based flow control) does not have this nice cushion to fall back on, and NACK-based AIMD schemes suffer a higher packet loss increase than equivalent ACK-based schemes. In the next section, we will look at the scalability of general binomial algorithms and study how we can reduce the amount of packet loss as the number of flows increases.

V. Packet-loss Scalability of Congestion Control

A. Overview

Suppose the average packet loss when n flows share a link of capacity C is given by p_n. Let *packet loss increase factor* s_n be the ratio of p_n to p_1. Parameter s_n specifies how fast packet loss increases when more flows share a common link and directly relates to the ability of the scheme to support a large number of flows (i.e., schemes with lower s_n scale better). Using (29), we derive:

$$p_n \approx \frac{\lambda^2 (k+2) n^{2k+2}}{C^{2k+2}\left(1-\left(1-\sigma(C/n)^{l-1}\right)^{k+2}\right)}, \qquad (31)$$

and using a two-term approximation from (27):

$$s_n \approx \frac{n^{l+2k+1}\left(2-(k+1)\sigma C^{l-1}\right)}{2-(k+1)\sigma(C/n)^{l-1}} = O(n^{l+2k+1}). \qquad (32)$$

Hence, packet loss increase factor s_n of binomial algorithms is proportional to n^{l+2k+1} for small n and grows no faster than n^{l+2k+1} for the rest of n. For AIMD, we get the familiar scalability formula of n^2, whereas the IIAD (i.e., $k = 1$, $l = 0$) and SQRT (i.e., $k = l = ½$) algorithms scale as n^3 and $n^{2.5}$, respectively. Furthermore, among all *TCP-friendly* schemes (i.e., $k + l = 1$), packet loss increase s_n is proportional to n^{3-l}, which means that TCP-friendly schemes with the *largest* l scale best. Since we already established that l must be no more than 1 (the non-cross-over condition), we arrive at our first major conclusion – *among TCP-friendly binomial schemes, AIMD scales best*.

We should make several observations about the applicability of (32) in practice. First, we assumed in (29) that the overshoot will be as large as possible, i.e., $\lambda U^{-k} R$. However, in many cases the actual overshoot will be some random value distributed between zero and $\lambda U^{-k} R$. Second, recall our discussion of AIMD's scalability in the previous section. When the increase distance $U–L$ becomes small compared to the value of the increase step, AIMD starts scaling as a linear function rather than a quadratic function. Hence, (32) is accurate only when the increase steps are small compared to C/n. The results based on the above model can be further skewed, if $\lambda U^{-k} R$ becomes large compared to X, in which case we must use the exact formula in (29).

B. Simulation

To verify these theoretical results and show some examples, we present simulation results of AIMD(1,½) and IIAD(1,½) schemes over a T1 link (i.e., C = 1,544 kb/s). For AIMD, we set *MTU/RTT* at two constant values of 5,000 and 50,000 bps (the corresponding schemes will be called $AIMD_1$ and $AIMD_2$) to show how their scalability changes when λ becomes large compared to the upper boundary $U = C/n$. For IIAD we selected *MTU/RTT* = 10,000 bps to allow the scheme to maintain $p_n << 100\%$ (otherwise, IIAD loses its n^3 packet loss increase). We used a discrete event simulation, in which n flows of the same type shared a common link. We used our prior assumption of immediate and synchronized feedback, as well as the assumption that the flows employed a NACK-based protocol (i.e., "open-loop" congestion control).

Figure 3 shows the variation of parameter s_n (based on the *actual*, rather than the *maximum* overshoot) during the simulation as a function of n for the three flows. In $AIMD_1$, packet loss increase ratio s_{100} reaches a factor of 6,755, which is equivalent to scalability of $n^{1.91}$ (just below the predicted n^2). On the other hand, $AIMD_2$ maintains its quadratic packet-loss increase only until $n = 7$, at which time it switches to a linear increase. The $AIMD_2$ scheme reaches an increase factor of $s_{100} = 352$, which is equivalent to an overall scalability of $n^{1.27}$. It may seem at first that the larger increase step λ of the $AIMD_2$ scheme is better; however, due to a larger λ, $AIMD_2$ is much more aggressive in searching for bandwidth and suffers a lot more packet loss than $AIMD_1$ for all values of n. Thus, for ex-

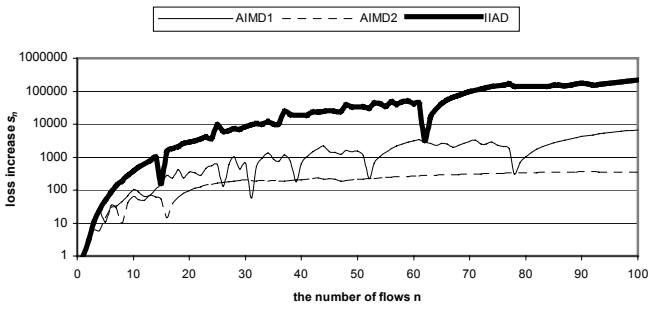

Figure 3. Parameter s_n (i.e., packet-loss scalability) of AIMD and IIAD in simulation based on actual packet loss.

ample, for $n = 100$, $AIMD_2$ loses 55% of all sent packets, while $AIMD_1$ loses only 10%.

Finally, IIAD's scalability performance is much worse than that of either of AIMD schemes as can be seen in the figure. The packet loss with 100 flows (i.e., p_{100}) is 219,889 times larger than the packet loss with one flow (i.e., p_1). Hence, under the given conditions, the overall scalability of IIAD is approximately $n^{2.67}$ (again slightly less than the predicted n^3).

As pointed out before and as shown in Figure 3, the *actual* increase in packet loss under non-ideal (i.e., discrete) conditions may be lower than that predicted by (32). Nevertheless, the theoretical result in (32) can be used as a good performance measure in comparing the scalability of different binomial schemes (e.g., as predicted, AIMD scales much better than IIAD).

C. Feasibility of Ideal Scalability

We next examine the "ideal scalability" of binomial schemes and derive its necessary conditions. We define a scheme to have *ideal scalability*, if s_n is constant for all n. This definition is driven by the fact that no matter how small packet loss p_1 can be made in a *non-scalable* scheme (i.e., a scheme with quickly-growing s_n), there will be a link of sufficient capacity that will accommodate such large number of concurrent flows n that p_n will be unacceptably high. This is especially true given the no-better-than-quadratic scalability of binomial congestion control. Consequently, the ideal situation would be to have a scheme that maintains a consistent packet loss rate regardless of the number of flows utilizing the scheme over a shared link, i.e., $p_n = p_1$ for all n. Furthermore, we would like to have a scheme that maintains the *same* packet loss over links of *different* capacity C.

To solve the above problem, we examine (32) again in order to find congestion control schemes that allow s_n to remain constant. Clearly, the necessary conditions for this ideal scalability are (the second condition is needed for convergence):

$$\begin{cases} l + 2k + 1 = 0 \\ k + l > 0 \end{cases} \Rightarrow \begin{cases} k < -1 \\ l > 1 \end{cases}. \tag{33}$$

The $l > 1$ condition means that if we plan to satisfy the non-cross-over conditions (16), (21), or prevent the scheme from reducing its rate below zero, ideal scalability requires the knowledge of some tight upper limit on sending rate x (see discussion following (21) earlier). Consequently, only assuming that x is limited by a constant (i.e., C), is it possible to find such σ that will satisfy the necessary condition $\sigma < 1/(lx^{l-1})$ in (21) for all rates $0 < x \le C$. Hence, we come to our second major conclusion – *among I-D congestion control schemes, ideal scalability is possible only when sending rates x are limited from above by a constant, i.e., when flows have the knowledge of the bottleneck capacity.*

There are two simple ways how an application can learn the value of C – by using real-time end-to-end measurements or by asking the network to provide an explicit feedback with the value of C. In the next section, we will examine the viability of applying the former method to sampling the capacity of the bottleneck link and the possibility of using such estimates in ideally-scalable congestion control.

Note that *all* flows sharing a single link must receive an estimate of C that is fairly close to the true capacity of the link[6]. A major drawback of employing congestion control that relies on real-time estimates of C is that different flows may form a different estimate, which may result in poor convergence and/or scalability depending on the amount of error. Hence, our approach in this section relaxes one condition (i.e., $l \le 1$), but imposes a new one – all flows must measure the bottleneck capacity with high *consistency*. Note that a thorough evaluation of various bandwidth estimation methods for the purpose of ideally-scalable congestion control is beyond the scope of this paper.

We also speculate that schemes with ideal scalability may be somewhat difficult to use in practice due to two factors – errors in measuring capacity C [6] and typically slower convergence to fairness due to less-aggressive probing for bandwidth. Nevertheless, we investigated ideally-scalable congestion control until we established a working version of the algorithm, which we will present in the remainder of the paper. Note that much more work in this area is required before we can recommend an I-D congestion control method other than AIMD for practical use over the Internet.

D. Ideally-Scalable Congestion Control

In this section, we introduce a new method, called *Ideally-Scalable Congestion Control* (ISCC), and show how values of the bottleneck capacity can be used to select the values of (λ,σ). Note that other ways of selecting (λ,σ) may be possible to achieve the same goal of constant s_n. We use notation ISCC(x) to refer to the ideally-scalable scheme described in this section with parameter l equal to x and parameter k equal to $-(l+1)/2$.

Assuming that C is known and assuming that $x(t) \le C$ at all times t (i.e., each application will limit its sending rate to be no higher than C), we can satisfy $\sigma < 1/(lx^{l-1})$ in (21) by choosing the following σ:

$$\sigma = \frac{1}{m_D C^{l-1}}, \tag{34}$$

where $l > 1$ and m_D is some constant greater than or equal to l. It is easy to show that the decrease step of schemes with σ according to (34) is no more than x/m_D for any given state

[6] The more the error, the slower will be the convergence. Unfortunately, the lack of space does not permit us to show this result more conclusively.

$x > 0$. Hence, rate x is guaranteed to stay positive at all times. By varying constant m_D, the scheme can adjust its average efficiency, where larger values of m_D mean higher efficiency.

In addition, we must carefully select the value of λ so that the negative value of power k is not allowed to cause uncontrollably-high increase steps. One way to attempt to achieve this is to select a fixed value α and then multiply it by $(MTU/RTT)^{k+1}$ as shown in (4). However, the increase steps will still remain virtually unlimited, because the value of *MTU/RTT* has little relationship to the value of C (which is needed to effectively limit λx^{-k}). In addition, different flows may use different multiplicative factors in (4) due to the differences in the RTT or the MTU. An alternative approach is to apply a similar thinking to that used before in selecting σ – choose λ so that the increase step is always no more than x/m_I for any given rate x, where m_I is some constant greater than or equal to one. This can be written as:

$$\lambda x^{-k} \le x / m_I, m_I \ge 1, \tag{35}$$

which is satisfied with the following choice of λ:

$$\lambda = \frac{C^{k+1}}{m_I}, m_I \ge 1. \tag{36}$$

Parameter m_I can be used to vary the aggressiveness of the scheme in searching for new bandwidth, where larger values of m_I result in less aggressive behavior of the scheme.

Furthermore, the above selection of λ and σ allows us to separate the value of packet loss p_1 from the capacity of the bottleneck link C. Combining (29), (34) and (36), we get that packet loss of ISCC schemes is *link-independent*:

$$p_1 = \frac{k+2}{m_I^2\left(1-\left(1-1/m_D\right)^{k+2}\right)+k+2} \tag{37}$$

In the next section, we compare the performance of one particular ISCC congestion control scheme in a NACK-based real-time streaming application with that of IIAD, AIMD, and TFRC (TCP-Friendly Rate Control) [7].

VI. Experiments

A. Choice of Powers Functions

We start with an observation that if l becomes much larger than 1.0 in an ISCC scheme and sending rate x is much smaller than capacity C (e.g., when n is large), such congestion control becomes less responsive to packet loss. Being less responsive usually results in very small rate reductions that often cannot elevate congestion in a single step. Thus, schemes with large l usually need multiple back-to-back decrease steps to move the system below the efficiency line in Figure 1. Our assumptions above do not model this behavior and the actual resulting packet loss in these schemes turns out to be higher than predicted by (32) and the convergence time is sometimes substantially increased.

Hence, from this perspective, larger values of l are not desirable. The only value of l that guarantees ideal scalability among *TCP-friendly* schemes (i.e., $k + l = 1$ and $l + 2k + 1 = 0$) is quite high and, specifically, equals 3. In practice, this scheme converges very slowly[7] and may not be a feasible solution for the real Internet. Among non-TCP-friendly schemes, values of l close to 1.0 force k to come close to –1.0 (because $l+2k+1$ must still remain zero), which also results in slower convergence to fairness as sum $k + l$ approaches zero[8].

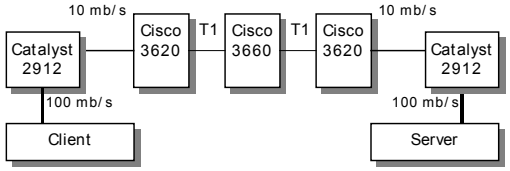

Figure 4. Setup of the experiment.

Among an infinite number of ISCC schemes, we arbitrarily selected a scheme with $l = 2$ ($k = –1.5$), which achieves reasonable performance in terms of both packet loss and convergence, and show its performance in this paper. Note that this particular scheme is somewhat less aggressive that TCP and typically would yield bandwidth to TCP, if employed over a shared path (however, this effect becomes noticeable only when the number of flows n is large). Hence, the practical application of this ISCC scheme in the Internet would require the use of new QoS methods in routers (i.e., DiffServ) as discussed in the introduction. Alternatively, it may be possible to use other ISCC schemes (with a different l), which are not penalized by TCP and which do not suffer from much slower convergence. Due to limited space, we consider finding the best ISCC scheme to be beyond the scope of this paper.

B. Real-time Bandwidth Estimation

In this section, we briefly examine the accuracy of real-time bandwidth estimation in our NACK-based streaming application and in the next section, we show the performance of ISCC(2), which relies on these real-time estimates for computing the values of λ and σ.

We used a Cisco network depicted in Figure 4 for all real-life experiments in this paper. During the experiment, we disabled WRED and WFQ on all T1 interfaces to reflect the current setup of backbone routers. The server supplied real-time bandwidth-scalable MPEG-4 video, which included the FGS (Fine-Granular Scalable) enhancement layer [17] and the regular base layer, to the client. Consequently, at any time t, the server was able to adapt its streaming rate to the rate $x(t)$ requested by the client, as long as $x(t)$ was no less than the rate of the base layer b_0 and no more than the combined rate of both layers.

We used a 10-minute MPEG-4 video sequence with the base layer coded at b_0 = 14 kb/s and the enhancement layer coded up to the maximum rate of 1,190 kb/s. Note that two concurrent flows were needed to fully load the bottleneck link. Hence, our experiments below do not cover the case of n = 1, and s_n is defined as the ratio of p_n to p_2.

During the experiment, the client applied a simple packet-bunch estimation technique [4], [16] to server's video packets. To simplify the estimation of the bottleneck bandwidth, the

[7] Slow convergence was found experimentally.

[8] Values of $k + l$ close to zero mean that the system makes very small steps toward the fairness line and thus, converges very slowly.

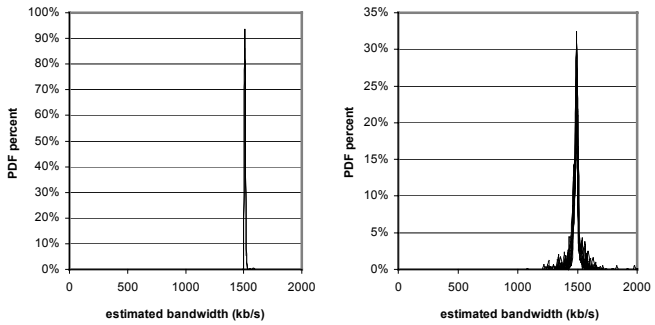

Figure 5. The PDFs of bandwidth estimates with 2 (left) and 32 (right) AIMD(1,½) flows over a shared T1 link.

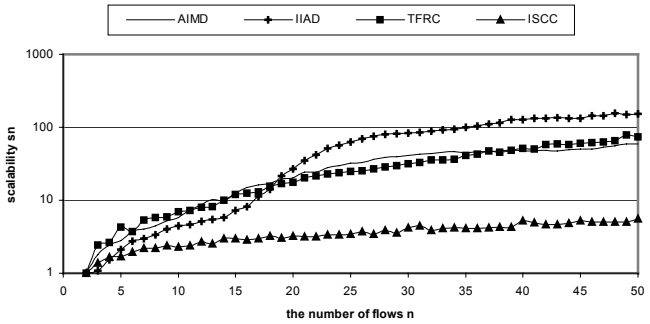

Figure 6. Packet-loss increase factor s_n for the Cisco experiment.

server sent its packets in bursts of a pre-defined length. A bandwidth sample was derived from each burst that contained at least three packets.

To establish a baseline performance, Figure 5 (left) plots the PDFs of IP bandwidth estimates[9] obtained by two AIMD(1,½) flows over the T1 link in Figure 4 (both flows used a fixed value of *MTU/RTT* equal to 30 kb/s). As the figure shows, the flows measured the IP bottleneck bandwidth to be 1,510 kb/s, which is very close to the actual T1 rate of 1,544 kb/s (the discrepancy is easily explained by the data-link/physical layer overhead on the T1 line). Furthermore, both flows were in perfect agreement, and 99.5% of estimates of each flow were between 1,500 and 1,520 kb/s.

Figure 5 (right) shows the PDFs of bandwidth estimates obtained by 32 simultaneous AIMD(1,½) flows running over the same topology in Figure 4 and with the same value of *MTU/RTT*. This time, the majority of estimates lie in the proximity of 1,490 kb/s, and 95.5% of estimates are contained between 1,400 and 1,620 kb/s (i.e., within 7% of 1,510 kb/s). The lower accuracy of bandwidth estimation in the second case is explained by the lower average sending rate of each flow (i.e., 36 kb/s compared to 559 kb/s in the first case).

Nevertheless, what matters most to the ISCC congestion control is the ability of flows to establish *consistent* estimates, rather than *accurate* estimates. To this extent, we found that the actual disagreement between the flows during the experiment was negligible and did not noticeably impact packet loss rates or fairness. Due to limited space, we skip a detailed performance study of our bandwidth estimation scheme and move on to show the scalability results in the next section.

C. Scalability Results

We extensively tested ISCC in simulation and found that it performed very well, in fact achieving constant packet loss. Due to the lack of space, we were forced to remove the simulation results from this paper, and show only real-life experiments over a Cisco testbed (see below).

In our application with NACK-based congestion control, all methods used slow start at the beginning of each transfer; however, the results below exclude the behavior of the network during slow start and focus on the performance of the schemes in the interval starting 5 seconds after the *last* flow finished its slow start and ending when the *first* flow terminated.[10] This interval was 520 to 600 seconds long (depending on the number of flows) and included a combined transfer of approximately 60,000 packets.

During the experiment, we tried to select the parameters of the schemes so that the average packet loss of two competing flows using each scheme was between 0.3% and 0.6%. This constraint resulted in selecting *MTU/RTT* equal to 30 kb/s for AIMD(1,½), and 50 kb/s for IIAD(½,2). The value of the *MTU variable* in TFRC's equation [7] was selected to be 180 bytes, whereas the *actual* MTU used during the experiment was 1,500 bytes for all schemes. Note that TFRC was the only protocol, which used real-time measurements of the RTT in its computation of the rate.

The efficiency and aggressiveness parameters of the ISCC(2) scheme were set with the same goal in mind to maintain low initial packet loss p_2: $m_D = 2$ and $m_I = 20$. These parameters guarantee that each flow does not decrease its rate by more than ½ and does not probe for new bandwidth more aggressively that by 5% (i.e., 1/20) of the current sending rate.

The results of the experiment are summarized in Figure 6, which shows packet-loss increase factor s_n for four different schemes and values of n between 2 and 50. The results of the experiment show that all non-scalable schemes maintained a steady packet-loss increase to well over 15%. For example, IIAD reached $p_{50} = 45\%$ (p_2 = 0.29%), AIMD 22% (p_2 = 0.38%), and TFRC 20% (p_2 = 0.26%).

On the other hand, the packet loss of the ISCC(2) scheme climbed only to 3.1% over the same range of flows n (p_2 = 0.57%). A least-squares fit suggests that the increase in ISCC's packet loss is very slow, but noticeable (i.e., $n^{0.47}$). Thus, even though the ISCC scheme was not able to achieve constant packet loss in practice, it did show a substantially better performance than any other scheme.

In addition, under the worst conditions (i.e., $n \approx 50$), our data show that the non-scalable protocols maintained a "frozen" picture between 11% and 42% of the corresponding session due to *underflow events* (which are produced when a frame is missing from the decoder buffer at the time of its decoding). Clearly, these results indicate that high packet loss is very harmful, even in the presence of low RTTs (50-200 ms), large startup delays (3 seconds in our case), and an efficient packet loss recovery mechanism (our retransmission scheme

[9] Note that bandwidth *estimates* were derived from bandwidth *samples* by using the median of the past 20-seconds worth of samples.

[10] Flows were started with a 1.5-second delay.

were able to recover all base-layer packets before their deadlines until loss rates exceeded approximately 15%).

At the same time, the ISCC(2) scheme was able to recover *all* frames (including base and enhancement layer) before their decoding deadlines, representing an ideal streaming situation for an end-user.

Therefore, we come to the conclusion that non-scalable schemes are poorly suited for rate-based protocols that do not utilize self-clocking and that ideally-scalable schemes promise to provide a constant packet-loss scalability not only in simulation, but also in practice. Nevertheless, further study is required in this area to understand the tradeoffs between the different values of l and k, as well as establish whether slower convergence to fairness found in simulation has any strong implications in large networks (i.e., in the real Internet).

VII. Conclusion

The difficulty of "open-loop" congestion control stems from the fact that the sender in such protocols is not governed by "self-clocking" of acknowledgements and typically continues to stress the network at the same rate even in the presence of severe packet loss and congestion. In such situations of aggravated packet loss, the main problem of NACK-based congestion control can be narrowed down to cases when the client either does not receive any server packets at all (which by default prohibits it from changing the server's rate), or takes multiple retransmissions of control messages to notify the server about the new reduced rate.

Interestingly, these problems are only noticeable when the congestion is severe enough to require multiple retransmissions of the client's control messages, or when the network encounters periods of *heavily-bursty* loss. Our experiments with traditional (i.e., non-scalable) NACK-based congestion control methods found that packet loss rates increased very rapidly as the number of flows on the shared link increased.

To investigate this observation further, we analyzed the class of binomial algorithms and derived the formulas of packet loss increase factor s_n as a function of the number of flows: $s_n = O(n^{l+2k+1})$. Using our derivations we found that among all proposed binomial schemes, AIMD had the best scalability $O(n^2)$ and the lowest packet loss. Furthermore, we showed that unless the schemes had the knowledge of bottleneck capacity C, the scalability of AIMD could not be improved, and even the performance of AIMD was inadequate for actual use in NACK-based applications. Even though all the derivations in the paper assumed synchronized and immediate feedback, our final formulas were found to hold in a number of streaming experiments over a real Cisco network with random packet loss and delayed feedback.

Given the knowledge of the bottleneck bandwidth, we showed that ideal scalability was both theoretically and practically possible; however, the ISCC schemes were found to be slower in their convergence to fairness when the number of flows n was large. Even though ISCC schemes are "more careful" in probing for new bandwidth, the average efficiency of these schemes was no worse than that of AIMD or IIAD (due to limited space, not discussed in the main body of the paper).

Regardless of whether ISCC is a viable protocol for the current or future (i.e., DiffServ) Internet, this paper not only answered the question of why NACK-based congestion control is "difficult," but it also measured the exact magnitude of this "difficulty" and provided one solution that overcomes the rapid packet-loss increase typical to "open-loop" congestion control.

References

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *IETF RFC 2581*, April 1999.

[2] D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms," *IEEE INFOCOM*, April 2001.

[3] D. Bansal, H. Balakrishnan, S. Floyd, S. Shenker, "Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms," *ACM SIGCOMM*, August 2001.

[4] R.L. Carter, and M.E. Crovella, "Measuring Bottleneck Link Speed in Packet Switched Networks," *International Journal on Performance Evaluation* 27 & 28, 1996.

[5] D-M. Chiu, and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol. 17, 1989.

[6] C. Dovrolis, P. Ramanathan, and D. Moore, "What Do Packet Dispersion Techniques Measure?" *IEEE INFOCOM*, April 2001.

[7] S. Floyd, M. Handley, and J. Padhye, "Equation-Based Congestion Control for Unicast Applications," *ACM SIGCOMM*, September 2000.

[8] S. Floyd, M. Handley, and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control," *ACIRI Technical Report* http://www.aciri.org/tfrc/aimd.pdf, May 2000.

[9] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM*, 1988.

[10] T. Kim, S. Lu, and V. Bharghavan, ``Loss Proportional Decrease based Congestion Control in the Future Internet," *University of Illinois Technical Report* http://timely.crhc.uiuc.edu/Drafts/tech.lipd.ps.gz, July 1999.

[11] K-W. Lee, T. Kim, V. Bharghavan, "A Comparison of End-to-End Congestion Control Algorithms: The Case of AIMD and AIPD," *University of Illinois Technical Report* http://timely.crhc.uiuc.edu/~kwlee/psfiles/infocom2001.ps.gz, 2000.

[12] K-W. Lee, R. Puri, T. Kim, K. Ramchandran, V. Bharghavan, "An Integrated Source Coding and Congestion Control Framework for Video Streaming in the Internet," *IEEE INFOCOM*, March 2000.

[13] A. Mena, and J. Heidemann, "An Empirical Study of Real Audio Traffic," *IEEE INFOCOM*, March 2000.

[14] T. Nandagopal, K-W. Lee, J.R. Li, V. Bharghavan, "Scalable Service Differentiation Using Purely End-to-End Mechanisms: Features and Limitations," *IFIP/IEEE International Workshop on Quality of Service (IWQoS)*, June 2000.

[15] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *ACM SIGCOMM*, September 1998.

[16] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," *Ph.D. dissertation*, UC Berkeley, 1997.

[17] H. Radha, Y. Chen, K. Parthasarathy, and R. Cohen, "Scalable Internet Video Using MPEG-4," *Signal Processing: Image Communication*, 1999.

[18] RealPlayer, Real Networks, http://www.real.com.

[19] R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-End Rate-based Congestion Control Mechanism for Real-time Streams in the Internet," *IEEE INFOCOM*, March 1999.

[20] Y.R. Yang, M.S. Kim, and S.S. Lam, "Transient Behaviors of TCP-friendly Congestion Control Protocols," *IEEE INFOCOM*, April 2001.

[21] Y.R. Yang and S.S. Lam, "General AIMD Congestion Control," *University of Texas at Austin Technical Report* ftp://ftp.cs.utexas.edu/pub/lam/gaimd.ps.gz, May 2000.