Retransmission Schemes for Streaming Internet Multimedia: Evaluation Model and Performance Analysis

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

This paper presents a trace-driven simulation study of two classes of retransmission timeout (RTO) estimators in the context of real-time streaming over the Internet. We explore the viability of employing retransmission timeouts in NACK-based (i.e., rate-based) streaming applications to support multiple retransmission attempts per lost packet. The first part of our simulation is based on trace data collected during a number of real-time streaming tests between dialup clients in all 50 states in the U.S. (including 653 major U.S. cities) and a backbone video server. The second part of the study is based on streaming tests over DSL and ISDN access links. First, we define a generic performance measure for assessing the accuracy of hypothetical RTO estimators based on the samples of the round-trip delay (RTT) recorded in the trace data. Second, using this performance measure, we evaluate the class of TCP-like estimators and find the optimal estimator given our performance measure. Third, we introduce a new class of estimators based on delay jitter and show that they significantly outperform TCP-like estimators in NACK-based applications with *low-frequency* RTT sampling. Finally, we show that high-frequency sampling of the RTT completely changes the situation and makes the class of TCP-like estimators as accurate as the class of delay-jitter estimators.

1. INTRODUCTION

Many Internet transport protocols rely on retransmission to recover lost packets. Reliable protocols (such as TCP) utilize a well-established sender-initiated retransmission scheme that employs retransmission timeouts (RTO) and duplicate acknowledgements (ACKs) to detect lost packets [9]. RTO estimation in the context of retransmission refers to the problem of predicting the next value of the round-trip delay (RTT) based on the previous samples of the RTT. RTO estimation is usually a more complicated problem than simply predicting the *most likely* value of the next RTT. For example, an RTO estimator that always underestimates the next RTT by 10% is significantly worse than the one that always overestimates the next RTT by 10%. Although both estimators are within 10% of the correct value, the former estimator generates 100% duplicate packets, while the latter one avoids all duplicate packets with only 10% unnecessary waiting.

Even though Jacobson's RTO estimator [9] is readily accepted by many TCP-like protocols, the problem of estimating the RTO in streaming protocols has not been addressed before. Current streaming protocols [22] deployed in the Internet rely on NACK-based flow control and usually do not implement congestion control, and the question of whether TCP's RTO estimator is suitable for such protocols remains an open issue. This paper sheds new light on the problem of RTO estimation in NACK-based protocols and shows the performance of several classes of RTO estimators in realistic Internet streaming scenarios.

Traditionally, NACK-based protocols sample the RTT only at times of packet loss (see below for details of how this is done). Even though there is nothing that inherently stops NACK-based protocols from sampling the RTT at a higher rate, our study for the most part follows the assumptions of the existing NACK-based applications [22] (i.e., the receiver sends messages to the server only upon packet loss and the RTT is measured only for the retransmitted packets).

As a result of our investigation, we found that TCP's RTO was an inadequate predictor of future values of the RTT when used in a NACK-based protocol over paths with *low-frequency* RTT sampling (i.e., low packet loss). We further found that along such paths, the accuracy of estimation could be substantially improved if the client used delay jitter in its computation of the RTO. On the other hand, when the RTT sampling rate was increased, TCP's RTO performed very well and the benefits of delay jitter were much less significant. Since an application typically does not know its future packet loss rates, we find that NACK-based protocols, augmented with high-frequency (i.e., in the order of once per RTT) sampling of the round-trip delay, will perform very well regardless of the end-to-end characteristics of a particular path (for example, high-

ACM Computer Communication Review, vol. 32, no. 2, April 2002.

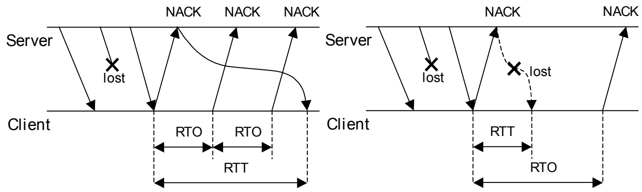

Figure 1. Underestimation results in duplicate packets (left) and overestimation results in unnecessary waiting (right).

frequency RTT sampling in real-time streaming can be implemented by using congestion control messages and once-per-RTT receiver-based feedback [13]).

In addition, this paper presents a generalized (i.e., suitable for many real-time applications) NACK-based, real-time retransmission model for multimedia streaming over the Internet and assess the effectiveness of various RTO estimators in the context of Internet streaming and our retransmission model. While the primary goal of our study is to develop a better retransmission mechanism for real-time applications, our retransmission model and new performance measure introduced in this paper are generic enough to apply to TCP as well.

Our characterization of RTO estimators is based on a reasonably large number of real-time streaming tests conducted between dialup clients from all 50 states in the U.S. and a backbone server during a seven-month period. We believe that this setup accurately reflects the current situation with real-time streaming in the Internet since the majority (i.e., 87-89%) of households in the U.S. still connect to the Internet through dialup modems [5], [8].

A good RTO estimator is the basis of any retransmission scheme. An application utilizing an RTO estimator that consistently *underestimates* the round-trip delay generates a large number of duplicate packets. The effect of duplicate packets ranges from being simply wasteful to actually causing serious network congestion. Note that in NACK-based applications, the receiver (i.e., the client) is responsible for estimating the RTO and the server is no longer in charge of deciding when to initiate a particular retransmission. This is illustrated in Figure 1 (left), in which the client sends three NACKs in response to a single lost packet and produces two duplicate packets due to insufficient RTO.

On the other hand, *overestimation* of the RTT defers the generation of subsequent retransmission requests and leads to lower throughput performance in TCP and causes an increased number of *underflow events* (which are generated by packets arriving after their decoding deadlines) in real-time applications. In either case, the amount of overestimation can be measured by the duration of unnecessary waiting for timeouts (i.e., waiting longer than the RTT of the lost retransmission). This is illustrated in Figure 1 (right). In the figure, the first retransmission is lost as well, and the generation of the second NACK is significantly delayed because the RTO is higher than the network RTT.

Therefore, the performance (i.e., accuracy) of an RTO estimator is fully described by two parameters (quantified later in this paper) – the number of duplicate packets and the amount of unnecessary timeout waiting. These two parameters cannot be minimized at the same time, since they represent a basic trade-off of any RTO estimator (i.e., decreasing one parameter will increase the other). To study the performance of RTO estimators, we define a weighted sum of these two parameters and study a multidimensional optimization problem in order to find the tuning parameters that make an RTO estimator optimal within its class. The minimization problem is not straightforward because the function to be minimized is non-continuous, has unknown (and often non-existent) derivatives, and contains a large number of local minima.

The paper is organized as follows. Section 2 provides the background on the problem of estimating retransmission timeouts and discusses some of the related work. Section 3 describes the methodology of our experiment. Section 4 introduces a novel performance measure that is used to judge the accuracy of hypothetical RTO estimators throughout this paper. Section 5 studies the class of TCP-like RTO estimators and models their performance. Section 6 discusses a new class of jitter-based RTO estimators and shows their superior performance in our modem datasets. Section 7 studies the performance of RTO estimators along high-speed Internet paths with high-frequency RTT sampling and shows that these paths require a different estimator. Section 8 concludes the paper.

2. BACKGROUND AND RELATED WORK

Recall that TCP's RTO estimation consists of three algorithms. The first algorithm, *smoothed RTT estimator* (SRTT), is an exponentially-weighted moving average (EWMA) of the past RTT samples [1], [9], [19]:

$$SRTT_i = \begin{cases} RTT_0, & i = 0 \\ (1-\alpha)\cdot SRTT_{i-1} + \alpha \cdot RTT_i, & i \geq 1 \end{cases}, \qquad (1)$$

where RTT_i is the i-th sample of the round-trip delay produced at time t_i and α (set by default to 1/8) is a smoothing factor that can be varied to give more or less weight to the history of RTT samples. In the original RFC 793 [20], the RTO was obtained by multiplying the latest value of the SRTT by a fixed factor between 1.3 and 2.0. In the late 1980s, Jacobson [9] found that the RFC 793 RTO estimator produced an excessive amount of duplicate packets when employed over the Internet and proposed that the second algorithm, *smoothed RTT variance estimator* (*SVAR*), be added to TCP's retransmission scheme [1], [9], [19]:

$$SVAR_i = \begin{cases} RTT_0/2, \ i = 0 \\ (1-\beta)\cdot SVAR_{i-1} + \beta \cdot VAR_i, \ i \ge 1 \end{cases}, \quad (2)$$

where β (set by default to ¼) is an EWMA smoothing factor and VAR_i is the absolute deviation of the i-th RTT sample from the previous smoothed average: $VAR_i = |SRTT_{i-1} - RTT_i|$. Current implementations of TCP compute the RTO by multiplying the smoothed variance by four and adding it to the smoothed round-trip delay [1], [19]:

$$RTO(t) = n \cdot SRTT_i + k \cdot SVAR_i, \quad (3)$$

where t is the time at which the RTO is computed, $n = 1$, $k = 4$, and $i = \max i$: $t_i \le t$.

The third algorithm involved in retransmission, *exponential timer backoff*, refers to Jacobson's algorithm [9] that exponentially increases the timeout value each time the same packet is retransmitted by the sender. Exponential timer backoff does not increase the accuracy of an RTO estimator, but rather conceals the negative effects of underestimating the actual RTT.[1] Since this paper focuses on tuning the accuracy of RTO estimators, we consider the timer backoff algorithm to be an orthogonal issue, to which we will not pay much attention. Furthermore, real-time applications have the ability to utilize a different technique that conceals RTT underestimation, which involves setting a deterministic limit on the number of retransmission attempts for each lost packet based on real-time decoding deadlines.

Rigorous tuning of TCP's retransmission mechanism has not been attempted in the past (possibly with the exception of [1]), and the study of TCP's RTO over diverse Internet paths is limited to [17], [18], in which Paxson found that 40% of retransmissions in the studied TCP implementations were redundant.[2]

Recently, Allman and Paxson [1] conducted a trace-driven simulation study based on TCP traffic to investigate the performance of hypothetical TCP-like RTO estimators (3) for several values of α, β, and k (n was kept at 1). The authors compared the performance of eight estimators by varying (α,β) and keeping k fixed at 4 and examined eight additional estimators by running k through eight integer values and keeping (α,β) fixed at their default values. The paper further concluded that no TCP-like RTO estimator could perform significantly better in the future versions of TCP than Jacobson's de-facto standard [9] and that even varying parameter n in (3) would not make the estimator substantially better.

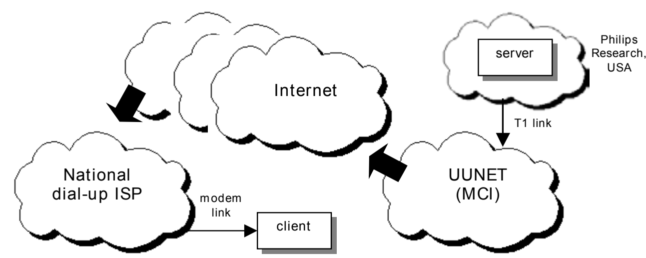

Figure 2. The setup of the modem experiment.

Among other reliable protocols with non-Jacobson RTO estimation, Keshav *et al.* [11] employed sender-based retransmission timeouts equal to twice the *SRTT* (i.e., the RFC 793 estimator), and Gupta *et al.* [7] used a NACK-based retransmission scheme, in which receiver timeouts and detection of lost packets were based on inter-packet arrival delay jitter.

The situation with RTO estimation in real-time streaming applications is somewhat different – the majority of real-time protocols either use TCP's RTO or rely on novel RTO estimation methods whose performance in the real Internet is unknown. Papadopoulos *et al.* [16] proposed a real-time retransmission scheme in which the receiver used the value of the SRTT in (1) to decide which packets were eligible for the first retransmission and employed special packet headers to support subsequent retransmissions. The benefit of avoiding timeouts was offset by the inability of the proposed scheme to overcome NACK loss. Rhee [23] employed a retransmission scheme in which the sender used three *frame durations* (instead of an estimate of the RTT) to decide on subsequent retransmissions of the same packet. A similar sender-based retransmission scheme was proposed by Gong *et al.* [6], with the exception that the sender used an undisclosed estimate of the RTT to decide when the same packet was eligible for a repeated retransmission.

3. METHODOLOGY

3.1. Experiment

Our evaluation study of RTO estimators is based on experimental data collected in a large-scale real-time streaming experiment over the Internet during November 1999 – May 2000. Aiming to create a setup the reflects the current use of real-time streaming in the Internet by the majority of home users [22], we implemented an MPEG-4 real-time streaming client-server architecture with NACK-based retransmission and used it to sample the RTT process along diverse paths in the dialup Internet.

To achieve an extensive coverage of dialup points in the U.S., we selected three major national dialup ISPs (which we call ISP_a, ISP_b, and ISP_c), each with at least five hundred V.90 (i.e., 56 kb/s) dialup numbers in the U.S. and several million active subscribers. We further designed an

[1] Another method of reducing the number of duplicate packets in TCP is to use a minimum of 1 second in (3), as suggested in a recent IETF document [19].

[2] Note that not all redundant retransmissions were due to an insufficient RTO.

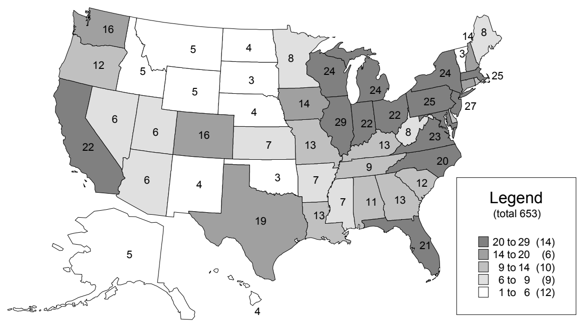

Figure 3. The number of cities per state that participated in the streaming experiment.

experiment in which hypothetical Internet users of all 50 states dialed a local access number to reach the Internet and streamed video sequences from our backbone server. Although the clients were physically located in our lab in the state of New York, they dialed long-distance phone numbers (see Figure 2) and connected to the Internet through a subset of the ISPs' 1813 different V.90 access points located in 1188 U.S. cities. A detailed description of the experiment can be found in [12].

We used two 10-minute QCIF (176x144) MPEG-4 sequences coded at video bitrates of 14 and 25 kb/s. The corresponding *IP bitrates* (i.e., including IP, UDP, and our streaming headers) were 16.0 and 27.4 kb/s, respectively.

During the experiment, the clients performed over 34 thousand long-distance phone calls and received 85 million packets (27.1 GBytes of video data) from the server. The majority of end-to-end paths between the server and the clients contained between 10 and 13 hops (with 6 minimum and 22 maximum). Moreover, server packets in our experiment traversed 5,266 distinct Internet routers, passing through 1003 dialup access points in 653 major U.S. cities (see Figure 3) [12].

3.2. RTT Measurement

In order to maintain an RTO estimator, the receiver in a real-time session must periodically measure the round-trip delay. In our experiment, the client obtained RTT measurements by utilizing the following two methods. The first method used packet loss to measure the round-trip delay – each successfully recovered packet provided a sample of the RTT (i.e., the RTT was the duration between sending a NACK and receiving the corresponding retransmission). In order to avoid the ambiguity of which retransmission of the same packet actually returned to the client, the header of each NACK request and each retransmitted packet contained an extra field specifying the retransmission attempt for that particular packet. Thus, the client was able to pair retransmitted packets with the exact times when the corresponding NACKs were sent to the server (i.e., Karn's [10] retransmission ambiguity problem was avoided).

The second method of measuring the RTT was used by the client to obtain *additional* samples of the round-trip delay in cases when network packet loss was too low. The method involved periodically sending *simulated* retransmission requests to the server if packet loss was below a certain threshold. In response to these simulated NACKs, the server included the usual overhead[3] of fetching the needed packets from the storage and sending them to the client. Note that even though we call these retransmissions "simulated," the round-trip delays they generated were 100% real and the use of these RTTs in updating the RTO estimator was fully justified. During the experiment, the client activated simulated NACKs, spaced 30 seconds apart, if packet loss was below 1%.

Note that all NACKs were sent using UDP, which made them susceptible to packet loss as well. Further discussion of the sampled RTTs, heavy-tailed distributions of the RTT, and various "sanity checks" can be found in [12].

4. PERFORMANCE

4.1. Retransmission Model

In real-time streaming, RTO estimation is necessary when the client supports multiple retransmission attempts per lost packet. After studying our traces, we found that 95.7% of all lost packets, which were recovered *before their deadline*, required a single retransmission attempt, 3.8% two attempts, 0.4% three attempts, and 0.1% four attempts. These results are important for two reasons.

First, 4.3% of all lost packets in our experiment could not be recovered with a single retransmission attempt. Even though it does not seem like a large number, our experiments with MPEG-4 indicate that there is no "acceptable" number of underflow events that a user of a real-time video application can feel completely comfortable with, and therefore, we believe that each lost packet must be recovered with as much *reasonable* persistence as possible.

Furthermore, since the average packet loss during the experiment was only 0.5% [12], the majority of retransmitted packets were able to successfully arrive to the client. However, in environments with a much lower end-to-end delay and/or higher packet loss[4], the percentage of packets recovered with a single retransmission attempt will be much lower than 95.7%. Besides the obvious higher probability of losing a retransmission or a NACK (due to higher packet loss), the RTT in such environments is likely to be much lower than the startup delay, which naturally allows more retransmission attempts per lost packet before the packet's deadline. Therefore, the existence of paths with lower de-

[3] Server logs showed that the overhead was below 10 ms for all retransmitted packets.

[4] For example, in certain DSL experiments with higher average packet loss, only 70% of the lost packets were recovered using one retransmission attempt.

lays and higher packet loss provides a strong justification for using more than one per-packet retransmission attempt in future streaming applications.

Second, our trace data show that if a lost packet in our experiment was successfully recovered *before its deadline*, the recovery was performed in no more than four attempts. The latter observation is used in our retransmission model (described later in this section) to limit the number of per-packet retransmission attempts (which we call R_{max}) to four. Note that this limit applies only to the collected traces and is not an inherent restriction of our model.

Ideally, an RTO estimator should be able to predict the exact value of the next round-trip delay. However, in reality, it is quite unlikely that any RTO estimator would be able to do that. Hence, there will be times when the estimator will predict smaller, as well as larger values than the next RTT. To quantify the deviation of the RTO estimate from the real value of the RTT, we utilize the following methodology.

Imagine that we sequentially number all *successfully recovered* packets in the trace (excluding simulated retransmissions) and let rtt_k be the value of the round-trip delay produced by the k-th successfully recovered packet at time t_k (see Figure 4). Note that we distinguish rtt_k from RTT_i, where the latter notation includes RTT samples generated by simulated retransmissions, and former one does not.

In Figure 4, the effective RTO for recovered packet k is computed at the time of the retransmission request, i.e., at time $t_{req}(k) = t_k - rtt_k$. Therefore, assuming that $RTO(t)$ is the value of the retransmission timeout at time t and assuming that the client uses the *latest* value of the RTO for each subsequent retransmission of a particular lost packet, it makes sense to examine how well the value of the RTO at the time of the request, $RTO(t_{req}(k))$, predicts the real value of the round-trip delay rtt_k. Hence, the accuracy of an RTO estimator in predicting the RTT of lost packets based on our trace data can be established by computing the *timeout waiting factor* w_k for each successfully recovered packet k in the trace:

$$w_k = \frac{RTO(t_k - rtt_k)}{rtt_k}. \tag{4}$$

Note that although our model does not use RTT samples measured by simulated retransmissions in computing w_k's (because they do not represent an actual loss), it uses them in updating the RTO estimator.

Since the exact effect of overestimation and underestimation of the RTT depends on whether the first retransmission of a particular packet was lost or not (and in some cases on whether subsequent retransmissions were lost or not), we simplify the problem and study the performance of RTO estimators assuming the worst case: values of w_k less than 1

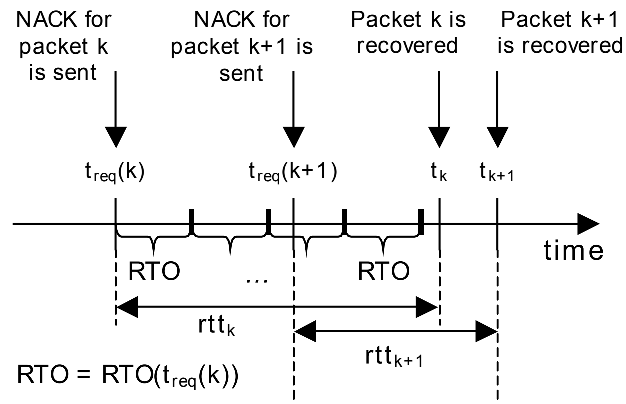

Figure 4. Operation of an RTO estimator given our trace data.

always indicate that the estimator would have tried (if not limited by R_{max}) to produce $\lfloor rtt_k / RTO(t_k - rtt_k) \rfloor = \lfloor 1/w_k \rfloor$ duplicate packets given our trace data (i.e., assuming that all retransmissions arrived to the client), and values of w_k greater than 1 always indicate that the estimator would have waited longer than necessary before detecting that a subsequent retransmission was needed (i.e., assuming that the first retransmission initiated at time $t_{req}(k)$ was lost). In Figure 4, given our assumptions, the RTO estimator generates four (i.e., $\lfloor 1/w_k \rfloor$) duplicate packets while recovering packet k.

The negative effects of duplicate packets (i.e., wasted bandwidth and aggravation of congestion) are understood fairly well. On the other hand, the exact effect of unnecessary timeout waiting in real-time applications depends on a particular video stream (i.e., the decoding delay of each frame), video coding scheme (i.e., the type of motion compensation, scalability, and transform used), individual lost packets (i.e., which frames they belong to), and the video startup delay.

Nevertheless, we can make a generic observation that RTO estimators with higher timeout overwaiting factors w_k suffer a lower probability of recovering a lost packet and consequently incur more underflow events. To keep our results universal and applicable to any video stream, we chose not to convert w_k's into the probability of an underflow event (or any other performance metric related to the video quality), and instead, study the tradeoff between a generic *average timeout overwaiting factor w* and the *percentage of duplicate packets d*:

$$w = \frac{1}{N_+} \sum_{w_k \ge 1} w_k \ , \tag{5}$$

$$d = \frac{1}{N} \sum_{w_k < 1} \min\left(\left\lfloor \frac{1}{w_k} \right\rfloor, R_{max} \right), \tag{6}$$

where N_+ is the number of times the RTO overestimated the next RTT (i.e., the number of times w_k was greater than or equal to 1) and N is the total number of lost packets. Parameter w is always above 1 and represents the average factor by which the RTO overestimates the RTT. Parameter d is the percentage of duplicate packets (relative to the number of lost packets) generated by the RTO estimator assuming that all requested retransmissions successfully arrived to the client.

In addition, we should note that the use of exponential backoff [5] instead of R_{max} provides similar, but numerically different results. However, in order to properly study the tradeoff between exponential backoff and R_{max}, our model must take into account retransmission attempts beyond the first one and study the probability of an underflow event in that context (i.e., the model must include a video coding scheme, video sequence, particular lost packets, and an actual startup delay). We consider such analysis to be beyond the scope of this paper.

Finally, we should point out that all RTO estimators under consideration in this paper depend on a vector of tuning parameters $\mathbf{a} = (a_1, \ldots, a_n)$. For example, the class of TCP-like RTO estimators in (3) can be viewed as a function of four tuning parameters α, β, k, and n. Therefore, the goal of the minimization problem that we define in the next section is to select such vector $\mathbf{a}$ that optimizes the performance of a particular RTO estimator $RTO(\mathbf{a}; t)$. By the word *performance* throughout this paper, we mean tuple (d,w) defined in (5) and (6).

4.2. Optimality and Performance

As we mentioned before, the problem of estimating the RTT is different from simply minimizing the deviation of the predicted value $RTO(\mathbf{a}; t_k - rtt_k)$ from the observed value rtt_k. If that were the case, we would have to solve a well-defined least-squares minimization problem (i.e., the maximum likelihood estimator):

$$\min_{(a_1,\ldots,a_n)} \sum_k \left(RTO(\mathbf{a}; t_k - rtt_k) - rtt_k\right)^2 . \tag{7}$$

The main problem with the maximum likelihood estimator (MLE) lies in the fact that the MLE cannot distinguish between over and underestimation of the RTT, which allows the MLE to assign equal cost to estimators that produce a substantially different number of duplicate packets.

Instead, we introduce two *performance functions* $\mathbf{H}(\mathbf{a})$ and $G(\mathbf{a})$ and use them to judge the accuracy of RTO estimators in the following way. We consider tuning parameter $\mathbf{a}_{opt}$ of an RTO estimator to be "optimal" within tuning domain S

[5] In which case, (6) should read $d = \dfrac{1}{N}\sum_{w_k<1}\left\lfloor \log_2\left(\dfrac{1}{w_k}+1\right)\right\rfloor$.

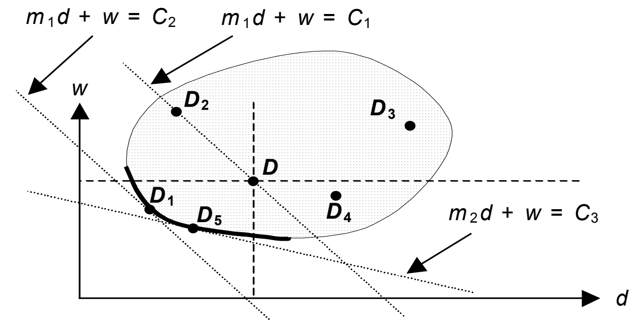

Figure 5. Comparison between RTO performance vector points (*d*,*w*).

of the estimator ($\mathbf{a}_{opt} \in S$), if $\mathbf{a}_{opt}$ minimizes the corresponding performance function (i.e., either $\mathbf{H}$ or G) within domain S. Later in this section, we will show that given the classes of RTO estimators studied in this paper and given our experimental data, the two performance measures (i.e., functions) produce equivalent results. Note that "optimality" is meaningful only within a given class of estimators, its tuning domain S, and the trace data used in the simulation.

In the first formulation, our goal is to minimize an *RTO performance vector-function* $\mathbf{H}(\mathbf{a}) = (d(\mathbf{a}),w(\mathbf{a}))$:

$$\min_{\mathbf{a}\in S} \mathbf{H}(\mathbf{a}) = \min_{\mathbf{a}\in S}\left(d(\mathbf{a}), w(\mathbf{a})\right). \tag{8}$$

For the minimization problem in (8) to make sense, we must also define vector comparison operators *greater than* and *less than*. The following are a natural choice:

$$(d_1,w_1) < (d_2,w_2) \Leftrightarrow \left((d_1 < d_2)\wedge(w_1 \le w_2)\right)\vee\left((d_1 \le d_2)\wedge(w_1 < w_2)\right), \tag{9}$$

$$(d_1,w_1) > (d_2,w_2) \Leftrightarrow \left((d_1 > d_2)\wedge(w_1 \ge w_2)\right)\vee\left((d_1 \ge d_2)\wedge(w_1 > w_2)\right), \tag{10}$$

and otherwise we consider tuples (d_1,w_1) and (d_2,w_2) to be *equivalent*. Figure 5 illustrates the above operators for a given RTO estimator and provides a graphical mapping between the performance of an RTO estimator and points on a 2-D plane. The shaded convex area in Figure 5 is the range of a hypothetical RTO estimator, where the range is produced by varying tuning parameter $\mathbf{a}$ within the estimator's tuning domain S (i.e., the convex area consists of points $\mathbf{H}(\mathbf{a})$, $\forall\mathbf{a}\in S$). Given a particular point $D = (d,w)$ in the range, points to the left and down from D (e.g., D_1) clearly represent a better estimator; points to the right and up from D (i.e., D_3) represent a worse estimator; and points in the other two quadrants may or may not be better (i.e., D_2 and D_4).

In order to help us understand which performance points in Figure 5 are optimal, we define the *optimal RTO curve* to be such points in the (d,w) space, produced by the RTO estimator, that are *less than or equal* to any other point produced by the RTO estimator, i.e., all points $(d_{opt},w_{opt}) = \mathbf{H}(\mathbf{a}_{opt})$, $\mathbf{a}_{opt}\in S$, such that $\forall\mathbf{a}\in S$: $\mathbf{H}(\mathbf{a}_{opt}) \le \mathbf{H}(\mathbf{a})$. In Figure

5, the optimal RTO curve is shown in bold along the left bottom side of the shaded area. Hence, finding the set of tuning parameters **a** that map to the optimal RTO curve is equivalent to solving the minimization problem in (8).

In the second formulation, we can state the problem of finding a better RTO estimator as that of minimizing a weighted sum of the percentage of duplicate packets d and the average overwaiting factor w (similar methods are frequently used in rate-distortion theory). The problem in the new formulation is easier to solve since it involves the minimization of a *scalar* function instead of a *vector* function. In addition, our reformulation allows us to decide on the exact relationship between *equivalent* points (i.e., in cases when neither (9) nor (10) holds) by assigning proper weight to one of the parameters in the (d,w) tuple.

Hence, we define a *weighted RTO performance function* $G(\mathbf{a}, M)$ as following:

$$G(\mathbf{a}, M) = M \cdot d(\mathbf{a}) + w(\mathbf{a}),\ 0 \le M < \infty\,, \qquad (11)$$

where M is a weight, which assigns desired importance to duplicate packets d (large M) or overwaiting factor w (small M). As we will see below, by running M through a range of values and optimizing $G(\mathbf{a}, M)$ for each weight M, we can build the optimal RTO curve; however, the actual values of M used to build the curve are not important.

Note that using performance function G we can unambiguously establish a relationship between *equivalent* points in the (d,w) space, given a certain weight M (i.e., points **a** with smaller $G(\mathbf{a}, M)$ are better). Specifically, for each weight M and for any constant $C > 0$, there exists a *performance equivalence* line $Md + w = C$, along which all points (d,w) are *equal* given the performance function in (11); points below the line are better (i.e., they belong to lines with smaller C); and points above the line are worse. In Figure 5, two parallel lines are drawn for $M = m_1$ using two different values of the constant ($C_2 < C_1$). Given weight m_1, point D_2 is now *equal* (not just equivalent) to D, point D_1 is still better, point D_3 is still worse, while point D_4 is now also worse.

In addition, not only is point D_1 better than D given performance function $G(\mathbf{a}, M)$ and weight m_1, but D_1 is also the "optimal" point of the RTO estimator in Figure 5 for weight m_1, i.e., point D_1 minimizes function (11) for weight m_1 within tuning domain S. In other words, to graphically minimize function $G(\mathbf{a}, M)$ for any weight M, one needs to slide the performance equivalence line $Md + w$ as far left and down as possible, while maintaining the contact with the range of the RTO estimator.

Notice how point D_1 found by minimizing function $G(\mathbf{a}, M)$ lies on the optimal RTO curve earlier defined using the performance measure in (8). We can further generalize this observation by saying that if the optimal RTO

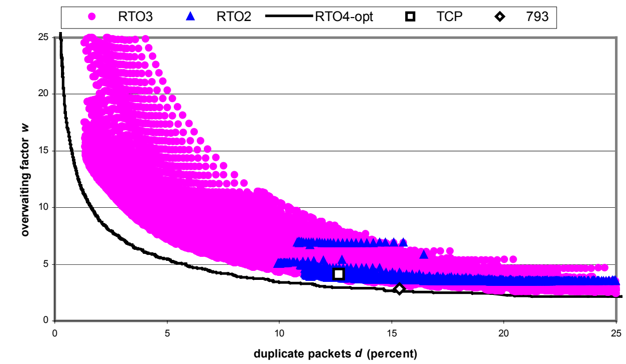

Figure 6. Performance of TCP-like estimators.

curve is given by a convex continuous function similar to the one in Figure 5, all points that optimize the weighted performance function $G(\mathbf{a}, M)$ will lie on the optimal RTO curve (and vice versa).

Consequently, using intuition, we can attempt to build the entire optimal RTO curve out of points $D_{opt}(M) = (d_{opt}(M), w_{opt}(M))$, where $d_{opt}(M)$ and $w_{opt}(M)$ are the result of minimizing $G(\mathbf{a}, M)$ for a particular weight M. For example, from Figure 5, we can conclude that optimal point $D_{opt}(m_1)$ is given by D_1 and optimal point $D_{opt}(m_2)$ is given by D_5. Hence, by varying M in $D_{opt}(M)$ between zero (flat performance equivalence line) and infinity (vertical performance equivalence line) we can produce (ideally) any point along the optimal RTO curve.

Note that we view the above retransmission model and both performance measures as an important contribution of this work. These techniques can be used to study the performance of RTO estimators in other datasets and even in ACK-based protocols (with properly taking into account exponential timer backoff as shown in section 4.1). The rest of the paper describes how our model and performance functions can be applied to the traces of our wide-scale Internet experiment [12] and discusses the important lessons learned.

Now we are ready to plot the values of vector function $\mathbf{H}(\mathbf{a})$ for different values of the tuning parameter $\mathbf{a} = (a_1, \ldots, a_n)$ in different RTO estimators, as well as identify the optimal points and understand which values of parameter **a** give us the best performance. Throughout the rest of the paper, in order to conserve space, we show the results derived from streaming traces through ISP_a (129,656 RTT samples). Streaming data collected through the other two ISPs produce similar results.

5. TCP-LIKE ESTIMATORS

5.1. Performance

We start our analysis of RTO estimators with a generalized TCP-like RTO estimator given in (3). We call this estimator RTO_4, because its tuning parameter **a** consists of four

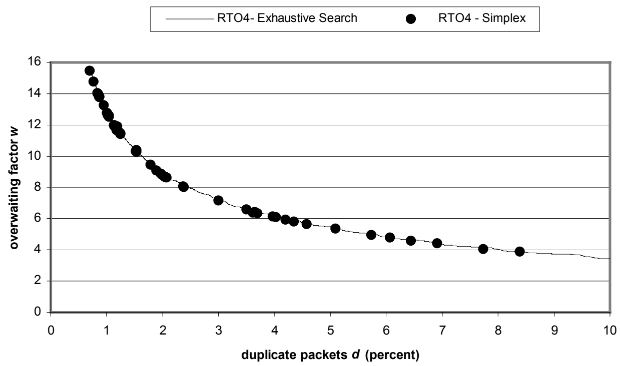

Figure 7. Points built by Downhill Simplex and the exhaustive search in the optimal RTO_4 curve.

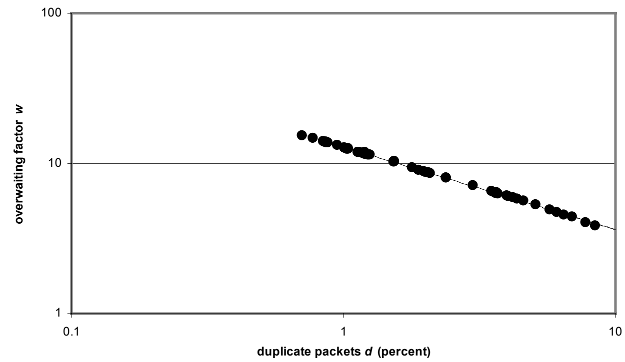

Figure 8. Log-log plot of the optimal (Simplex) RTO_4 curve.

variables: $\mathbf{a} = (\alpha, \beta, k, n)$. Recall that $\mathbf{a}_{TCP} = (0.125, 0.25, 4, 1)$ corresponds to Jacobson's RTO [9] and $\mathbf{a}_{793} = (0.125, 0, 0, 2)$ corresponds to the RFC 793 RTO [20].

In order to properly understand which parameters in (3) contribute to the improvements in the performance of the TCP-like estimator, we define two *reduced* RTO estimators depending on which tuning parameters (α, β, k, n) are allowed to vary. In the first reduced estimator, which we call RTO_2, we use only (α,β) to tune its performance, i.e., $\mathbf{a} = (\alpha, \beta, 4, 1)$. In the second reduced estimator, which we call RTO_3, we additionally allow k to vary, i.e., $\mathbf{a} = (\alpha, \beta, k, 1)$.

Figure 6 shows the optimal RTO_4 curve and the range of values $\mathbf{H}(\mathbf{a})$ produced by both reduced estimators. The ranges of RTO_2 (900 points) and RTO_3 (29,000 points) were obtained by conducting a uniform exhaustive search of the corresponding tuning domain S, and the optimal RTO_4 curve was obtained by extracting the minimum values of $\mathbf{H}(\mathbf{a})$ after a similar exhaustive search through more than 1 million points. In addition, Figure 6 shows the performance of Jacobson's RTO estimator, $\mathbf{H}(\mathbf{a}_{TCP}) = (12.63\%, 4.12)$, by a square and the performance of the RFC 793 RTO estimator, $\mathbf{H}(\mathbf{a}_{793}) = (15.34\%, 2.84)$, by a diamond. Clearly, Jacobson's and the RFC 793 RTO estimators are equivalent, since neither one is located below and to the left of the other.

The performance of RTO estimators in Figure 6 certainly gets better with the increase in the number of free tuning variables. For a given average overwaiting factor $w = 4.12$, RTO_2 and RTO_3 both achieve optimality in the same point and offer only a slight improvement in the number of duplicate packets over TCP RTO – 11.15% compared to 12.63%. RTO_4, however, offers a more substantial improvement, generating only $d = 7.84\%$ duplicate packets.

Furthermore, Figure 6 shows that the optimal RTO_4 curve (built by the exhaustive search) is convex and fairly continuous until approximately 20% duplicate packets. Consequently, we can build another optimal RTO_4 curve using scalar weighted performance function $G(\mathbf{a})$ and compare the results with those in Figure 6. A scalar function such as $G(\mathbf{a})$ allows us to use various numerical multidimensional minimization methods, which usually do not work with vector functions. In addition, we find that numerical optimization methods produce points along the optimal RTO curve with more accuracy than the exhaustive search (given a reasonable amount of time) and with fewer computations of functions $d(\mathbf{a})$ and $w(\mathbf{a})$ (i.e., faster).

To verify that weighted performance function $G(\mathbf{a})$ does in fact produce the same optimal RTO_4 curve, we focused on the following minimization problem for a range of values of weight M:

$$\min_{\mathbf{a}\in S} G(\mathbf{a}, M) = \min_{\mathbf{a}\in S}\big(M \cdot d(\mathbf{a}) + w(\mathbf{a})\big) \tag{12}$$

The fact that function $G(\mathbf{a}, M)$ has unknown (and non-existent) partial derivatives $\partial G(\mathbf{a}, M)/\partial a_k$ suggests that we are limited to numerical optimization methods that do not use derivatives. After applying the Downhill Simplex Method in Multidimensions (due to Nelder and Mead [15]) and quadratically convergent Powell's method [2], we found that the former method performed significantly better and arrived at (local) minima in fewer iterations. To improve the found minima, we discovered that restarting the Simplex method in random locations in the N-dimensional space ten times per weight M produced very good results.

Figure 7 shows the points built by the Downhill Simplex method for the RTO_4 estimator (each point corresponds to a different weight M) and the corresponding optimal RTO_4 curve previously derived from the exhaustive search. As the figure shows, points built by Downhill Simplex are no worse (and often slightly better) than those found in the exhaustive search.

Interestingly, the optimal curves in Figure 7 resemble power functions in the form of:

$$w_{opt} = C(d_{opt})^{-p}, p > 0. \tag{13}$$

To investigate this observation further, Figure 8 replots the points of the Downhill Simplex curve from Figure 7 on a

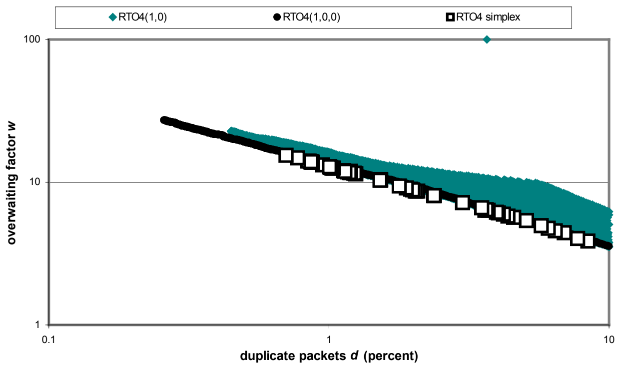

Figure 9. ***RTO***$_4$**-Simplex and two reduced** ***RTO***$_4$ **estimators on a log-log scale.**

log-log scale with a straight line fitted to the points. A straight line provides an excellent fit (with correlation 0.99) and suggests that the optimal RTO curve could be modeled as a power function (13) with $C = 1.022$ and $p = 0.55$.

Assuming that the relationship between w and d in the optimal RTO_4 curve is a power function (13), we can now analytically compute optimal points (d_{opt},w_{opt}) that minimize function $G(\mathbf{a})$ for a given weight M. Rewriting (11) using the function from (13), taking the first derivative, and equating it to zero we get:

$$\frac{\partial G(\mathbf{a})}{\partial d_{opt}} = \frac{\partial}{\partial d_{opt}}\left(Md_{opt} + Cd_{opt}^{-p}\right) = M - Cpd_{opt}^{-p-1} = 0\,. \quad (14)$$

Solving (14) for d_{opt} and using (13) one more time, we can express the optimal values of both the number of duplicate packets d_{opt} and the average overwaiting factor w_{opt} as a function of weight M:

$$d_{opt} = \left(\frac{Cp}{M}\right)^{\frac{1}{p+1}} \text{ and } w_{opt} = \frac{M}{p}\left(\frac{Cp}{M}\right)^{\frac{1}{p+1}}. \quad (15)$$

5.2. Tuning Parameters

In this section, we provide a reverse mapping from optimal performance points $\mathbf{H}(\mathbf{a})$ in Figure 7 to points $\mathbf{a}$ in tuning domain S (i.e., describe how to construct optimal RTO_4 estimators). While analyzing RTO_2, we noticed that for each given β, larger values of α produced fewer duplicate packets, as well as that for each fixed value of α, smaller values of β similarly produced fewer duplicate packets. To further study this phenomenon, we examined the correlation between the RTO_2 estimates and the corresponding future round-trip delays rtt_k for different values of (α,β). Interestingly, the highest correlation was reached in point (1.0, 0.044), which suggests that an RTO estimator with (α,β) fixed at (1,0) should provide estimates with a reasonably high correlation with the future RTT, as well as that it could be possible to achieve the values of the optimal RTO_4 curve by just varying parameters n and k in RTO_4.

To investigate this hypothesis, we constructed another reduced estimator called $RTO_{4(1,0)}$, which is produced by RTO_4 at input points (1, 0, k, n). The results of an exhaustive search of the reduced tuning domain (k, n) for $RTO_{4(1,0)}$ are plotted in Figure 9 (lightly shaded area). As the figure shows, the optimal RTO_4 curve (shown as squares in Figure 9) touches the range of $RTO_{4(1,0)}$, which means that the reduced estimator can achieve the points along the optimal RTO_4 curve while keeping α and β constant. This fact implies that it is not necessary to maintain a smoothed RTT average to achieve optimality within our datasets, because $\alpha = 1.0$ means that the *SRTT* always equals the latest RTT sample.

The next logical step is to question the need for *SVAR* in RTO_4 since *SVAR* turns out to be a constant when β equals zero. In the same Figure 9, we plotted an additional optimal curve for estimator $RTO_{4(1,0,0)}$, which represents RTO_4 at input points (1, 0, 0, n). As the figure shows, all values of the $RTO_{4(1,0,0)}$ estimator lie next to the optimal curve as opposed to many sub-optimal points produced by $RTO_{4(1,0)}$. At the end of this section, we discuss the explanation of why smoothing of RTT samples does not increase the accuracy of RTO_4, but first show how to construct an $RTO_{4(1,0,0)}$ estimator with a given performance.

A straight line fitted to the $RTO_{4(1,0,0)}$ curve in Figure 9 produces a power function (13) with $C = 1.07$ and $p = 0.546$. Further investigation discovered that there is a strong linear dependency between the optimal value of n_{opt} in $RTO_{4(1,0,0)}$ and the optimal value of the average overwaiting factor w_{opt}:

$$n_{opt} = mw_{opt} + b, \quad (16)$$

where $m = 0.86$ and $b = -0.13$. Since we already know the dependency between w_{opt} and d_{opt} in (13), we can derive the relationship between n_{opt} and d_{opt} in $RTO_{4(1,0,0)}$:

$$n_{opt} = mC(d_{opt})^{-p} + b. \quad (17)$$

Consequently, (17) can be used to build optimal $RTO_{4(1,0,0)}$ estimators given any desired value of duplicate packets d_{opt}. For example, if an application specifies that the maximum number of duplicate packets it is willing to tolerate is d_{opt} = 2%, using (13), the optimal overwaiting factor w_{opt} is 9.12 (the corresponding weight M is 248) and using (17), the optimal RTO estimator is given by $RTO_{4(1,0,0)}$ with n_{opt} = 7.31.

5.3. Discussion

This is the point when we must address a major conceptual difference between ACK and NACK-based retransmission schemes, as well as point out several properties of our experiment. The difference between ACK and NACK-based protocols lies in the fact that NACK-based applications obtain RTT samples only upon packet loss, while ACK-based applications consistently obtain RTT samples on a per-packet basis. Consequently, the distance between RTT samples in a NACK-based application is often large and fluctuates widely (i.e., between tens of milliseconds and tens of seconds). Given a low average packet loss of 0.5% during our Internet experiment, the average distance between consecutive RTT samples in our datasets was 15.7 seconds.

Hence, we observed that NACK-based protocols in the presence of low packet loss greatly undersample the RTT process, and further smoothing of already rare RTT samples with EWMA formulas produces a very sluggish and slow-responding moving average. Such moving average in the form of (1) and (2) can rarely keep up with the actual RTT and turns out to be a poor predictor of the future values of the round-trip delay. This observation represents the first major conclusion of our study – *NACK-based protocols in our experiment combined with low-frequency RTT sampling (i.e., low packet loss) required a different RTO estimation method than the classical Jacobson's RTO; specifically, smoothed averaging of RTT samples proved to be hurtful, and the latest RTT sample turned out to be the best predictor of the future RTTs.*

6. JITTER-BASED ESTIMATORS

6.1. Structure and Performance

The second class of RTO estimators, which we call RTO_J, is derived from $RTO_{4(1,0,0)}$ by adding to it a smoothed variance of the inter-packet arrival delay (quantified later in this section). As we will show below, RTO_J reduces the number of duplicate packets in our trace data compared to RTO_4 by up to 60%.

The receiver in a real-time protocol usually has access to a large number of delay jitter samples between the times when it measures the RTT. It would only be logical to utilize tens or hundreds of delay jitter samples between retransmissions to fine-tune RTO estimation. This fine-tuning is receiver-oriented and is not available to TCP senders (which they do not need since TCP obtains a substantial amount of RTT samples through its ACK-based operation). In fact, TCP's ability to derive an RTT sample from (almost) each ACK gave it an advantage that may now be available to NACK-based protocols in the form of delay jitter.

Before we describe our computation of delay jitter, we must introduce the notion of a packet burst. In practice, many real-time streaming servers are implemented to transmit their data in bursts of packets [14], [21] instead of sending one packet every so many milliseconds. Although the latter is considered to be an ideal way of sending video

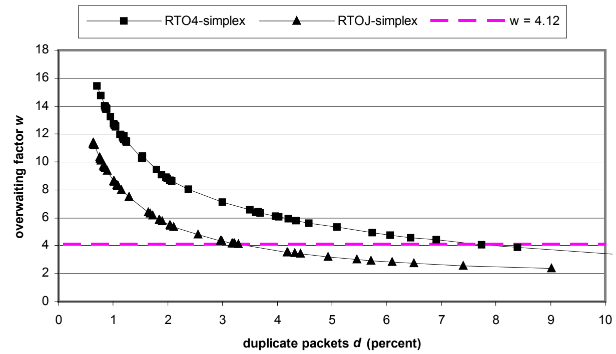

Figure 10. Jitter-based RTO estimators compared with the RTO_4 estimator.

traffic by many researchers (e.g., [4]), in practice, there are limitations that do not allow us to follow this ideal model [12].

In our server, we implemented bursty streaming with the burst duration D_b (i.e., the distance between the first packets in successive bursts) varying between 340 and 500 ms depending on the streaming bitrate (for comparison, RealAudio servers use D_b = 1,800 ms [14]). Each packet in our real-time application carried a burst identifier, which allowed the receiver to distinguish between packets from different bursts. After analyzing the traces, we found that *inter-burst* delay jitter had more correlation with the future RTT than *inter-packet* delay jitter (we speculate that one of the reasons for this was that more cross traffic was able to queue between the bursts than between individual packets).

To be more specific, suppose for each burst j, the last packet of the burst arrived to the client at time $t_{last}(j)$, and the first packet of the burst arrived at time $t_{first}(j)$. Consequently, the *inter-burst delay* for burst j is defined as:

$$\Delta_j = t_{first}(j) - t_{last}(k), j \geq 1 \tag{18}$$

where burst k is the last burst received before burst j (unless there is packet loss, $k = j - 1$). For each burst, using EWMA formulas similar to those in TCP, we compute *smoothed inter-burst delay* $S\Delta_j$ and *smoothed inter-burst delay variance* $SVAR\Delta_j$:

$$S\Delta_j = \begin{cases} \Delta_1, & j = 1 \\ (1-\alpha_1)\cdot S\Delta_{j-1} + \alpha_1 \cdot \Delta_j, & j \geq 2 \end{cases} \tag{19}$$

and

$$SVAR\Delta_j = \begin{cases} \Delta_1 / 2, & j = 1 \\ (1-\beta_1)\cdot SVAR\Delta_{j-1} + \beta_1 \cdot VAR\Delta_j, & j \geq 2 \end{cases}, \tag{20}$$

where α_1 and β_1 are exponential weights, and $VAR\Delta_j$ is the absolute deviation of Δ_j from its smoothed version $S\Delta_{j-1}$. In our experience, $S\Delta_j$ is usually proportional to burst duration

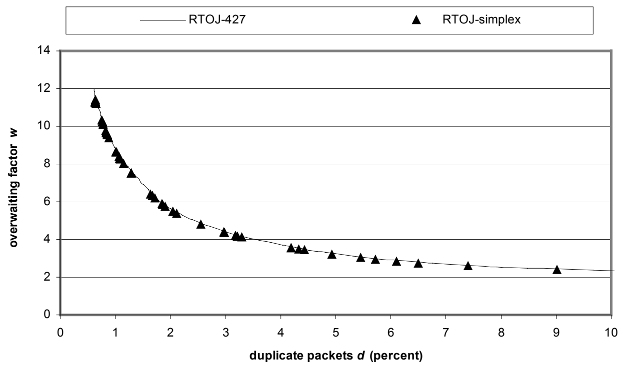

Figure 11. Reduced jitter-based estimator compared with the optimal RTO_J estimator.

D_b and thus, cannot be used the same way in real-time applications with different burst durations. On the other hand, smoothed variance $SVAR\Delta_j$ is fairly independent of the burst duration and reflects the variation in the amount of cross traffic in router queues along the path from the server to the client.

Given our definition of delay variation in (20), suppose that t_i is the time when our trace recorded the i-th RTT sample (including simulated retransmissions), then the effective *jitter-based* RTO at time t is:

$$RTO_J(t) = n \cdot RTT_i + m \cdot SVAR\Delta_j, \tag{21}$$

where $i = \max i$: $t_i \le t$ and $j = \max j$: $t_{first}(j) \le t$.

Figure 10 compares the performance of the RTO_J estimator with that of RTO_4 (both optimal curves were built using the Downhill Simplex method). Given a particular value of the average overwaiting factor w, RTO_J offers a 45-60% improvement over RTO_4 in terms of duplicate packets. Recall that for an average overwaiting factor $w = 4.12$, Jacobson's RTO estimator produced 12.63% duplicate packets and RTO_4 achieved 7.84%. At the same time, RTO_J is now able to improve this value to 3.25%.

6.2. Tuning Parameters

RTO_J contains four tuning variables $\mathbf{a} = (\alpha_1, \beta_1, m, n)$, just like the RTO_4 estimator. This time, however, the performance of the estimator does not strongly depend on the first two variables. Several values in the proximity of $\alpha_1 = 0.5$ give optimal performance. For β_1, the optimal performance is achieved at $\beta_1 = 0.125$, which is helpful if $SVAR\Delta_j$ is to be computed using only integer arithmetics. Just as in the $RTO_{4(1,0,0)}$ estimator, (α_1, β_1) can be fixed at their optimal values and the optimal RTO_J curve can be entirely built using n and m.

To further reduce the number of free variables in jitter-based estimators, we examined the relationship between

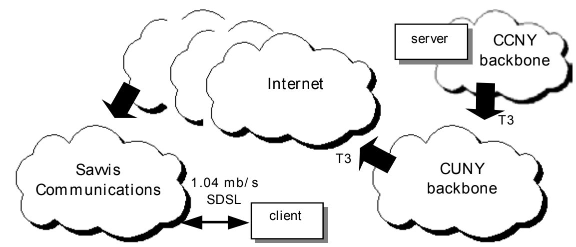

Figure 12. The setup of the high-speed experiment.

n_{opt} and m_{opt} in the optimal RTO_J curve shown in Figure 10. Although the relationship is somewhat random, there is an obvious linear trend, which fitted with a straight line (with correlation $\rho = 0.88$) establishes that function

$$m_{opt} = 4.2792 \cdot n_{opt} - 2.6646 \tag{22}$$

describes the optimal parameters n and m reasonably well. Consequently, we created a reduced estimator, which we call RTO_{J427}, by always keeping m as a function of n shown in (22) and compared its performance (by running n through a range of values) to that of RTO_J in Figure 11. As the figure shows, the reduced estimator RTO_{J427} reaches the corresponding optimal RTO_J curve with high accuracy.

Similar power functions (13) and (17) apply to the optimal RTO_J and RTO_{J427} curves. Table 1 summarizes the values of constants in both equations (13) and (17).

Table 1. Summary of constants in various power laws

Part I. Power function for optimal RTO curves: $w_{opt} = C(d_{opt})^{-p}$.			
RTO estimator	**C**	**p**	**correlation**
RTO_4	1.02	0.5500	0.9994
$RTO_{4(1,0,0)}$	1.07	0.5456	0.9991
RTO_J	0.50	0.6158	0.9997
RTO_{J427}	0.53	0.6098	0.9991
Part II. Power function for optimal parameter n: $n_{opt} = C_1(d_{opt})^{-p} + C_2$.			
Reduced estimator	**C_1**	**C_2**	**p**
$RTO_{4(1,0,0)}$	0.88	–0.13	0.5456
RTO_{J427}	0.20	0.31	0.6098

Using the same example from section 5, for $d_{opt} = 2\%$, we find that w_{opt} is 5.75 in RTO_{J427} (compared to 9.12 in $RTO_{4(1,0,0)}$). Given parameters in the second half of Table 1, the value of n_{opt} in RTO_{J427} is 2.47 (compared to 7.31 in $RTO_{4(1,0,0)}$), and the value of m_{opt} using (22) is 7.91. As we can see, the superior performance of the RTO_{J427} estimator over RTO_4 and $RTO_{4(1,0,0)}$ is achieved by placing lower weight on RTT samples and deriving more information about the network from the more frequent delay jitter samples.

Hence, we can summarize our second major conclusion as following – *during the experiment, a NACK-based RTO estimator running over paths with low-frequency RTT sampling (over 10 seconds between samples) could be signifi-*

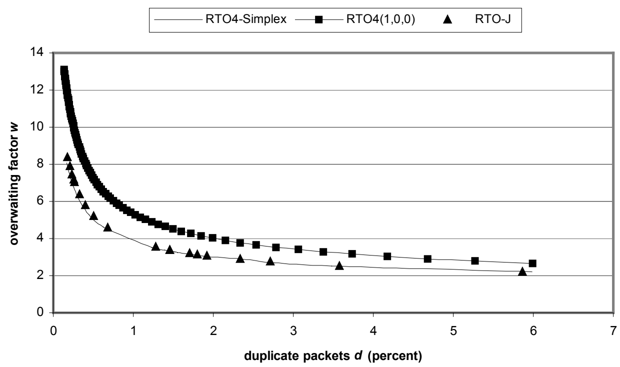

Figure 13. Performance of RTO_4, RTO_J and $RTO_{4(1,0,0)}$ in the CUNY dataset.

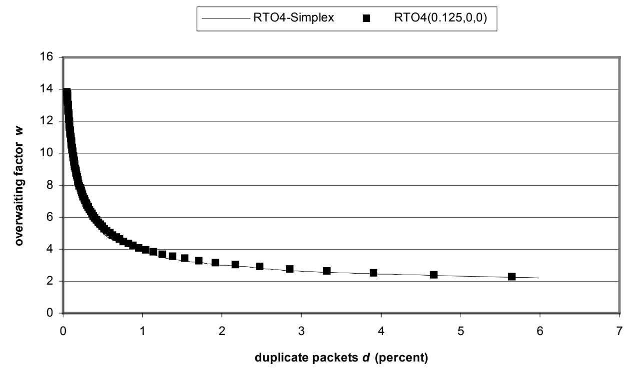

Figure 14. Performance of RTO_4 and $RTO_{4(0.125,0,0)}$ in the CUNY dataset.

cantly improved by adding smoothed delay jitter to the scaled value of the latest RTT.

7. HIGH-FREQUENCY SAMPLING

The final question left to resolve is whether the performance of RTO_4 and RTO_J is different in environments with high-frequency RTT sampling. In NACK-based protocols, high-frequency RTT sampling comes either from high packet loss rates or from frequent congestion control messages exchanged between the client and the server (in the latter case, the frequency of sampling is approximately equal to one sample per RTT [4]).

This section investigates the performance of RTO_4 and RTO_J in several environments with high-frequency RTT sampling and verifies whether the conclusions reached in previous sections hold for such Internet paths. We only show the results based on trace data collected along a single Internet path; however, the observations made in this section were also verified along multiple other paths with relatively high packet loss, as well as in a *congestion-controlled* streaming application with once-per-RTT sampling of the round-trip delay.

In this section, we apply trace-driven simulation to the datasets collected between a symmetric DSL (SDSL) client and a video server placed at the City College of New York (CCNY) during December 2000. This setup is shown in Figure 12. The CCNY backbone connected to the Internet through the CUNY (City University of New York) backbone via a series of T3 links. The client's dedicated SDSL circuit operated at 1.04 mb/s in both directions. The end-to-end path between the client and the server contained 15 routers from five Autonomous Systems (AS). During the experiment, we used a video stream coded at the video bitrate of 80 kb/s (86 kb/s IP bitrate). The collected dataset contains traces of 55 million packets, or 26 GBytes of data, obtained during the period of three weeks.

One interesting property of this end-to-end path is that the CUNY border router dropped a large fraction of packets during this experiment, regardless of the time of day or the sending rate of our flows. Thus, the average packet loss rate recorded in this trace was substantially higher than in the modem experiment (i.e., 7.4% vs. 0.5%), and the average delay between obtaining new RTT samples was only 740 ms, which is by a factor of 20 less than that in the wide-scale modem experiment.

Figure 13 shows the performance of the three estimators studied earlier in this paper in the CUNY dataset. All three optimal curves were built using Downhill Simplex. As the figure shows, both RTO_4 and RTO_J achieve the same optimal performance, which means that the addition of delay jitter to already-frequent RTT samples is not as beneficial as previously discovered. In addition, note that $RTO_{4(1,0,0)}$ is no longer optimal within the dataset. Both results were expected, because the higher sampling frequency in the CUNY dataset allows RTO_4 to be a much better predictor than it was possible in the modem datasets.

The final question that stands is what values of tuning variable **a** make RTO_4 optimal along paths with high-frequency RTT sampling (i.e., the CUNY dataset)? Our analysis of the data shows that variance estimator *SVAR* is still redundant and that RTO_4 can be reduced to a simpler estimator, which this time assumes the following form: $\mathbf{a} = (\alpha, 0, 0, n)$. Downhill Simplex optimization of RTO_4 shows that values of α between 0.12 and 0.13 are equally optimal and produce an estimator with performance equal to that of RTO_4. Note that Jacobson's value of $\alpha = 0.125$ falls within this range and agrees with the results derived from the CUNY dataset.

To verify that the reduced estimator $RTO_{4(0.125,\,0,\,0)}$ performs as well as RTO_4, we plotted both optimal RTO curves in Figure 14, which shows that the reduced estimator is almost identical to RTO_4.

Note that the above observations about the optimality of $RTO_{4(0.125,\,0,\,0)}$ were also found to hold when the CUNY server was replaced with a server located at Michigan State University, 21 hops from the client (the experiment was conducted in January 2001 and involved the transfer of over 17 million packets). Furthermore, similar results were obtained in various streaming tests over ISDN (over 77 million packets): in low packet-loss scenarios, RTO_J was significantly better than RTO_4, and $RTO_{4(1,0,0)}$ was optimal within the class of TCP-like estimators; however, in high packet loss scenarios, RTO_J did not offer much improvement over RTO_4.

We finish this section by reaching our third major conclusion – *in our experiments, along paths with high-frequency RTT sampling, a simple smoothed round-trip delay estimator SRTT with parameter α between 0.12 and 0.13 was the optimal estimator, and neither delay jitter nor delay variance estimator SVAR provided any added benefits.*

8. CONCLUSION

Current real-time streaming applications [22] rely on NACK-based retransmission and often do not implement congestion control. The nature of NACK-based retransmission and the lack of congestion control in these applications suggest that the current RTO estimation methods implemented in TCP may not be adequate for NACK-based streaming protocols. Furthermore, even the existing RTO estimation methods in TCP do not have a rigorous performance evaluation model, and their performance over diverse Internet paths remains unknown.

Our study introduced a novel performance measure (suitable for both NACK and ACK-based protocols), which captures the accuracy of hypothetical RTO estimators based on packet data traces of real Internet connections. This performance measure shows the inherent tradeoff between the number of duplicate packets generated by an estimator and the amount of unnecessary waiting for timeouts given the data traces.

Based on our performance measure, we found that in the dialup Internet (which is often accompanied by low packet loss [12]), TCP-like estimators were not optimal in traditional NACK-based protocols due to the large distance between RTT samples. We further established that in the dialup Internet, the latest RTT had the most relevance to the future RTTs and that EWMA smoothing of either RTT samples or RTT variance did not increase the accuracy of estimation. This result suggests that large distances between RTT samples (in the order of 15 seconds) allow end-to-end network conditions in the Internet to change significantly, which leads us to conclude that sampling rates of higher frequency may be required to adequately sample the RTT in the current Internet.

Furthermore, we found that frequent delay jitter samples were very useful in fine-tuning the RTO estimation between the measurements of the RTT. Delay-jitter estimators were found to perform much better in the modem traces; however, their benefits were virtually nullified when the RTT was sampled at a higher frequency.

As our results show, the performance of TCP-like estimators depends on the sampling frequency of the RTT. Consequently, we conclude that there is enough evidence to suggest that the paradigm in which NACK-based applications sample the RTT only at times of packet loss may not be very useful. We find that higher-frequency sampling of the RTT may be necessary for accurate RTO estimation and could be additionally used for other purposes (such as equation-based congestion control [4]). Our experiments with NACK-based congestion control show that RTT sampling rates of once-per-RTT can be achieved with very little overhead (i.e., the measurement of the RTT can be incorporated into the congestion control feedback loop). In such scenarios, our study found that a scaled *SRTT* estimator was optimal and even TCP's RTO was sufficiently accurate.

9. REFERENCES

[1] M. Allman and V. Paxson, "On estimating end-to-end network parameters," *ACM SIGCOMM*, August 1999.

[2] R.P. Brent, "Algorithms for Minimization without Derivatives," *Englewood Cliffs*, NJ, Prentice Hall, 1973.

[3] B. Dempsey, J. Liebeherr, and A. Weaver, "On retransmission-based error control for continuous media traffic in packet-switching networks," *Computer Networks and ISDN Systems*, vol. 28, no. 5, March 1996.

[4] S. Floyd, M. Handley, and J. Padhye, "Equation-based congestion control for unicast applications," *ACM SIGCOMM*, August 2000.

[5] S.P. Pizzo, "Why Is Broadband So Narrow?" *Forbes ASAP*, September 2001, p. 50.

[6] F. Gong and G.M. Parulkar, "An application-oriented error control scheme for high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 4, no. 5, 1996.

[7] R. Gupta, M. Chen, S. McCanne, and J. Walrand, "WebTP: a receiver-driven web transport protocol," *University of California at Berkeley Technical Report*, 1998.

[8] ISP Planet and Telecommunications Research International, "U.S. Residential Internet Market Grows in Second Quarter," http://isp-planet.com/research/2001/us_q2.html, 2001.

[9] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM*, 1988.

[10] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable protocols," *ACM SIGCOMM*, 1987.

[11] S. Keshav and S.P. Morgan, "SMART retransmission: performance with overload and random loss," *IEEE INFOCOM*, March 1997.

[12] D. Loguinov and H. Radha, "End-to-End Internet Video Traffic Dynamics: Statistics Study and Analysis," *IEEE INFOCOM*, July 2002.

[13] D. Loguinov and H. Radha, "Increase-Decrease Congestion Control for Real-time Streaming: Scalability," *IEEE INFOCOM*, July 2002.

[14] A. Mena and J. Heidemann, "An empirical study of real audio traffic," *IEEE INFOCOM*, March 2000.

[15] J.A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, 1965, pp. 308-313.

[16] C. Papadopoulos and G.M. Parulkar, “Retransmission-based error control for continuous media applications,” *IEEE NOSSDAV*, 1996.
[17] V. Paxson, “Measurements and analysis of end-to-end internet dynamics,” *Ph.D. dissertation*, Computer Science Department, University of California at Berkeley, 1997.
[18] V. Paxson, “End-to-End Internet Packet Dynamics,” *ACM SIGCOMM*, September 1997.
[19] V. Paxson and M. Allman, “Computing TCP's retransmission timer,” *IETF RFC 2988*, November 2000.
[20] J. Postel, “Transmission control protocol – DARPA Internet program protocol specification,” *IETF RFC 793*, September 1981.
[21] H. Radha, Y. Chen, K. Parthasarathy, and R. Cohen, “Scalable internet video using MPEG-4,” *Signal Processing: Image Communication*, 1999.
[22] Real Player, Real Networks. http://www.real.com.
[23] I. Rhee, “Error control techniques for interactive low bitrate video transmission over the Internet,” *ACM SIGCOMM*, August 1998.