On Retransmission Schemes for Real-time Streaming in the Internet

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

Hayder Radha [**]
[**]Philips Research
345 Scarborough Road
Briarcliff Manor, NY 10510
hayder.radha@philips.com

***Abstract* – This paper presents a trace-driven simulation study of three classes of retransmission timeout (RTO) estimators in the context of low-bitrate real-time streaming over the Internet. We explore the viability of employing retransmission timeouts in NACK-based real-time streaming applications that support multiple retransmission attempts per lost packet. In such applications, real-time RTO estimation plays a major role (i.e., poor RTO estimation results in a larger number of duplicate packets and sometimes more frequent underflow events). Our study is based on trace data collected during a number of real-time streaming tests conducted between our dialup clients in all 50 states of the U.S. (including 653 major U.S. cities) and our backbone video server during a seven-month period. First, we define a generic performance measure for assessing the quality 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, find the most optimal estimator given our performance measure, and establish power laws that describe the tradeoff between the optimal number of duplicate packets and the optimal timeout waiting time. Third, we introduce a new class of RTO estimators based on delay jitter and show that they perform significantly better than TCP-like estimators in NACK-based applications. Finally, we gain a major insight into the RTT process by establishing which tuning parameters of an RTO estimator make it optimal given our performance measure and our experimental data, and give our explanation of the observed phenomena.**

I. 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)[1] to detect lost packets [7]. 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.

Recall that TCP's RTO estimation consists of three algorithms. The first algorithm, *smoothed RTT estimator* (SRTT), is an exponential-weighted moving average (EWMA) of the past RTT samples [1], [7]:

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

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 [17], 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 [7] 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], [7]:

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

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 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]:

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

where t is the time at which the RTO is computed, $n = 1$, $k = 4$, and $i = \max i$: $t_i \leq t$.[2]

The third algorithm involved in retransmission, *exponential timer backoff*, refers to Jacobson's algorithm [7] 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. 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.

[1] Duplicate ACKs are used in TCP's fast retransmit to infer packet loss and distinguish the latter from packet reordering. TCP's triple-ACK mechanism is more related to estimating the reordering delay than to estimating the RTT, and consequently, we consider it to be outside the scope of this paper.

[2] Note that the latest IETF RFC [16] on TCP RTO introduced slight changes to the above algorithm. The changes include a minimum of one second in (3).

Unfortunately, thorough tuning of TCP's retransmission mechanism has not been attempted in the past (possibly with the exception of [1]), and the attempts to document the behavior of TCP's RTO estimator over diverse Internet paths are limited to [15], in which Paxson found that 40% of retransmissions in the studied TCP implementations were redundant. Nevertheless, TCP's retransmission scheme seems to be automatically accepted by numerous protocols and, in essence, has remained unquestioned by the Internet community since its introduction 13 years ago.

Recently, Allman and Paxson [1] conducted a simulation based on TCP traces to study the performance of hypothetical TCP-like RTO estimators (3) for several different 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 future versions of TCP than Jacobson's de-facto standard [7] and that even varying parameter n in (3) would not make the estimator substantially better.

Among other *reliable* protocols with non-Jacobson RTO estimation, Keshav *et al*. [9] employed sender-based retransmission timeouts equal to twice the *SRTT* (i.e., the RFC 793 estimator) and Gupta *et al*. [6] used a NACK-based retransmission scheme, in which receiver timeouts and detection of lost packets were based on inter-packet arrival delay jitter.

On the contrary, both ACK and NACK-based real-time streaming applications do not possess a common (i.e., agreed-upon) retransmission scheme that is shown to perform well under heterogeneous Internet conditions. In fact, many proposed real-time streaming schemes do not specify the choice of an RTO estimator [3], [4], [13], do not deal with real-time decoding deadlines of individual frames, ignore the probability of packet reordering [3], [14], [19], and often neglect to set the limit on the maximum number of retransmission requests (where the limit could be based on the lost packet's decoding deadline, some fixed integer number, or both).

Papadopoulos *et al*. [14] 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 [19] 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*. [5], 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.

In this paper, we present 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 low-bitrate Internet streaming and our retransmission model. While the primary goal of our study is to develop a better retransmission mechanism for real-time applications, the methodology and performance measure used to judge the quality of different RTO estimators 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. (including 653 major cities) and a backbone server during a seven-month period.

Furthermore, we argue that a good RTO estimator is the basis of any retransmission scheme. An application utilizing an RTO estimator that consistently *underestimates* the round-trip delay is bound to generate a large number of duplicate packets. The effect of duplicate packets ranges from being simply wasteful to actually causing serious network congestion. Consequently, any real-time application with a wide-scale deployment in mind, must possess a good RTO estimator (besides a good congestion control scheme, which is outside the scope of this paper) that guarantees a low number of duplicate packets.

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 hypothetical RTT of the lost retransmission).

Therefore, the performance (i.e., quality) 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.

Before we begin, we must point out two limitations of our study. First, the results presented in this paper were obtained from a trace-driven simulation, rather than from live experiments with each RTO estimator in the Internet (which would be unrealistic to conduct given the fact that we analyzed several million RTO estimators) or from the application of a mathematical model of the Internet's RTT process to our NACK-based retransmission scheme. Second, we study only three classes of RTO estimators, and potentially, there could exist a more optimal estimator that does not belong to the three studied classes. Nevertheless, we believe that the results

presented in this paper are novel and play an important role in building future NACK-based applications.

The paper is organized as follows. Section II describes the methodology that we used to collect the experimental data. Section III introduces our performance measure that we used to judge the quality of hypothetical RTO estimators based on the trace data. Section IV studies the class of TCP-like RTO estimators and models their performance. Section V briefly examines the class of percentile-based RTO estimators and shows their suboptimality. Section VI discusses our new class of jitter-based RTO estimators and shows their superior performance in NACK-based retransmission schemes. Section VII concludes the paper.

II. Methodology

A. Experiment

Our evaluation study of RTO estimators is based on experimental data collected during November 1999 – May 2000. We implemented an MPEG-4 real-time streaming client-server architecture with simple NACK-based retransmission and set our goal to collect a large data set documenting the behavior of the RTT and delay jitter processes along diverse Internet paths. For that purpose, we selected three major national dialup ISPs (which we call ISP_a, ISP_b, ISP_c), each with at least five hundred V.90 (i.e., 56 Kbps) dialup numbers in the U.S., and designed an experiment in which hypothetical Internet users of all 50 U.S. 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 and connected to the Internet through a subset of the ISPs' 1813 different V.90 access points located in 1188 U.S. cities.

During the experiment, we used two 10-minute QCIF (176x144) MPEG-4 sequences coded at *ideal* (i.e., video) bitrates of 14 and 25 Kbps (which corresponded to *IP bitrates* of 16.0 and 27.4 Kbps respectively). Both sequences were coded at 5 frames per second (fps), and the size of the former video sequence was 1.05 MBytes, while the size of the latter one was 1.87 MBytes.

Furthermore, we divided a 7-day week into 56 three-hour timeslots (i.e., 8 timeslots per day) and made sure that each of the 50 states was *successfully* tested at least once during each timeslot of the week (by the word "successfully" we mean that at least one of each state's dialup numbers was able to sustain end-to-end streaming for 10 minutes at the target IP bitrate during the corresponding timeslot).

Our seven-month experiment resulted in transporting (excluding lost packets) of 85 million packets and 27.1 GBytes of video data from our backbone server to dialup clients in all 50 states of the U.S. Moreover, the end-to-end paths between the server and the clients included 5,266 distinct Internet routers and 1003 dialup access points in 653 major U.S. cities (see Figure 1).[3]

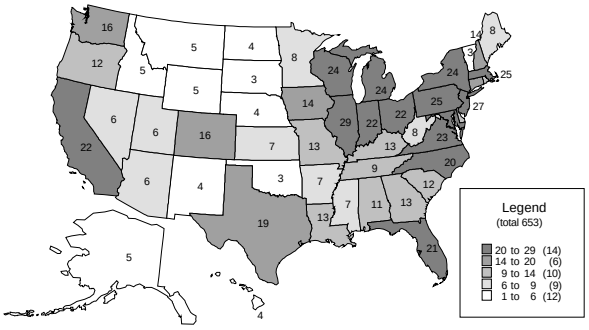

Figure 1. The number of cities per state that participated in the streaming experiment.

B. 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 number of the packet. Thus, the client was able to pair each retransmitted packet with the exact time when the corresponding NACK was sent out.

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[4] of fetching the needed packets from the storage and sending them to the client. During the experiment, the client activated simulated NACKs, spaced 30 seconds apart, if packet loss was below 1%.

III. Performance of RTO Estimators

A. Retransmission Model

In real-time streaming, RTO estimation is needed only if the client supports multiple retransmissions of the same 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,

[3] A detailed description of the experiment is available in a separate paper [10], which discusses both the design of the experiment and the behavior of observed network parameters (such as packet loss, RTT, delay jitter, packet reordering, path asymmetry, underflow events, video startup delays, etc.).

[4] The server overhead was below 10 ms for all retransmitted packets.

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. Therefore, if a real-time application employs a single retransmission per lost packet, it is bound to suffer 43 underflow events for each 1000 lost packets under similar streaming conditions. 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.

Second, our trace data show that if a lost packet is successfully recovered *before its deadline*, the recovery is done 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.

Given the fact that a substantial fraction of lost packets in our experiment required multiple retransmissions, we next focus on hypothetical RTO estimators that support multiple retransmission attempts per lost packet and define their performance based on our trace data. 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 method.

Imagine that we sequentially number all *successfully recovered* packets in the trace (excluding simulated retransmissions) and let $RTT_{rec}(i)$ be the value of the round-trip delay produced by the i-th successfully recovered packet at time $t_{rec}(i)$ (see Figure 2). The effective RTO for this lost packet would have been computed at the time of the retransmission request, i.e., at time $t_{req}(i) = t_{rec}(i) - RTT_{rec}(i)$. 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, $RTO(t_{req}(i))$, predicts the real value of the round-trip delay $RTT_{rec}(i)$. 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_i for each successfully recovered packet i in the trace:

$$w_i = \frac{RTO(t_{rec}(i) - RTT_{rec}(i))}{RTT_{rec}(i)}. \qquad (4)$$

In addition, we should note that although our model does not use RTT samples produced by simulated retransmissions in computing w_i's, it uses them in updating the RTO estimator.

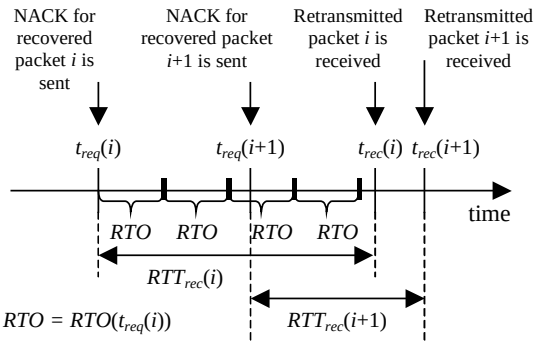

Figure 2. Operation of an RTO estimator given our trace data.

Since the exact effect of overestimation and underestimation of the RTT depends on whether the first retransmission 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_i less than 1 always indicate that the estimator would have tried (if not limited by R_{max}) to produce $\lfloor RTT_{rec}(i) / RTO(t_{rec}(i) - RTT_{rec}(i)) \rfloor = \lfloor 1/w_i \rfloor$ duplicate packets given our trace data (i.e., assuming that all retransmissions arrived to the client), and values of w_i 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 $t_{req}(i)$ was actually lost). In Figure 2, given our assumptions, the RTO estimator would have produced four (i.e., $\lfloor 1/w_i \rfloor$) duplicate packets while recovering packet i.

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_i are bound to suffer a lower probability of recovering a lost packet and consequently, more frequent underflow events. To keep our results universal and applicable to any video stream, we chose not to convert w_i's into the probability of an underflow event, and instead, we study the tradeoff between a generic *average timeout overwaiting factor* w and the percentage of duplicate packets d:

$$w = \frac{1}{N_+}\sum_{w_i \geq 1} w_i \quad \text{and} \quad d = \frac{1}{N}\sum_{w_i < 1} \min\left(\left\lfloor \frac{1}{w_i} \right\rfloor, R_{max}\right), \qquad (5)$$

where N_+ is the number of times the RTO overestimated the next RTT (i.e., the number of times w_i was greater than or equal to 1) and N is the total number of lost packets. Parameter w is always above 1 and provides an 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 in (5) instead of R_{max} provides similar, but numerically different results.[5] 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 our minimization problem is to select such vector **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).

B. Optimality and Performance

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

$$\min_{(a_1,\ldots,a_n)} \sum_i \left(RTO(\mathbf{a}; t_{rec}(i) - RTT_{rec}(i)) - RTT_{rec}(i)\right)^2 . \quad (6)$$

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 quality 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 of the estimator ($\mathbf{a}_{opt} \in S$), if $\mathbf{a}_{opt}$ minimizes the corresponding performance function (i.e., either **H** or G) within domain S. Furthermore, 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.

In the first formulation, we introduce a generic problem of minimizing an *RTO performance vector-function* $\mathbf{H}(\mathbf{a}) = (d(\mathbf{a}), w(\mathbf{a}))$:

$$\min_{\mathbf{a}\in S} \mathbf{H}(\mathbf{a}) = \min_{\mathbf{a}\in S} \big(d(\mathbf{a}), w(\mathbf{a})\big) . \quad (7)$$

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

$$(d_1, w_1) < (d_2, w_2) \Leftrightarrow \big((d_1 < d_2) \wedge (w_1 \le w_2)\big) \vee \big((d_1 \le d_2) \wedge (w_1 < w_2)\big), \quad (8)$$

$$(d_1, w_1) > (d_2, w_2) \Leftrightarrow \big((d_1 > d_2) \wedge (w_1 \ge w_2)\big) \vee \big((d_1 \ge d_2) \wedge (w_1 > w_2)\big), \quad (9)$$

and otherwise we consider tuples (d_1, w_1) and (d_2, w_2) to be *equivalent*. Figure 3 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 3 is the range of a hypothetical RTO estimator, where the range is produced by varying tuning parameter **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 of the RTO estimator, 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).

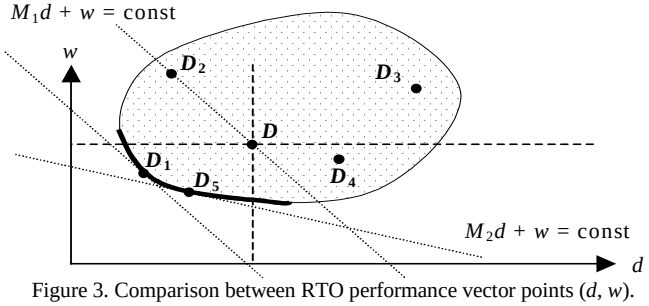

Figure 3. Comparison between RTO performance vector points (d, w).

In order to help us understand which performance points are most 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 $\forall \mathbf{a} \in S$: $(d_{opt}, w_{opt}) \le \mathbf{H}(\mathbf{a})$. In Figure 3, 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 produce the optimal RTO curve for a given RTO estimator is equivalent to solving the minimization problem in (7).

Alternatively, we can formulate 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. The problem in the new formulation appears 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 (8) nor (9) holds) by assigning proper weight to one of the parameters in the (d, w) tuple.

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

[5] For exponential backoff, (5) would read $d = \frac{1}{N} \sum_{w_i < 1} \left\lfloor \log_2\left(\frac{1}{w_i} + 1\right) \right\rfloor$.

$$G(\mathbf{a}) = M \cdot d(\mathbf{a}) + w(\mathbf{a}),\ 0 \le M < \infty, \qquad (10)$$

where M is a weight, which assigns desired importance to duplicate packets *d* (large *M*) or overwaiting factor *w* (small *M*). In practice, we cannot identify a value of weight *M* that is globally suitable for all real-time applications; however, we can now unambiguously establish a relationship between *equivalent* points in the (*d*, *w*) space for any given weight *M*. 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 (10); points below the line are better (i.e., they belong to lines with smaller *C*); and points above the line are worse. In Figure 3, two parallel lines are drawn for $M_1 = 1$ and different values of *C*. Given weight $M_1 = 1$, 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 $G(\mathbf{a})$ and weight M_1, but D_1 is also the *most optimal* point of the RTO estimator in Figure 3 for weight M_1 (i.e., D_1 is the solution to the minimization problem in (10) for weight M_1). In other words, to graphically minimize function $G(\mathbf{a})$ for any weight *M*, one needs to slide the performance equivalence line $Md + w$ as far left and down as possible, while maintaining contact with the range of the RTO estimator (in Figure 3, this procedure was performed for weight M_1, which resulted in finding point D_1). Notice how at the same time point D_1 lies on the optimal RTO curve defined using the performance measure in (7).

We can further generalize this observation by saying that if the optimal RTO curve is given by a convex continuous function similar to the one in Figure 3, all points that optimize the weighted performance function $G(\mathbf{a})$ 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 $G(\mathbf{a})$ for a particular weight *M*. For example, from Figure 3, we can conclude that the optimal point $D_{opt}(M_1)$ is given by D_1 and the 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.

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 using stream S_1 (129,656 RTT samples). The streaming data collected through the other two ISPs produce similar results.

IV. TCP-LIKE ESTIMATORS

A. Performance

We start our analysis of RTO estimators with a generalized TCP-like RTO estimator, which is given in (3) and which we call RTO_4. In RTO_4, tuning parameter **a** consists of four variables: $\mathbf{a} = (\alpha, \beta, k, n)$. Recall that $\mathbf{a}_{TCP} = (0.125, 0.25, 4, 1)$ corresponds to Jacobson's RTO [7] and $\mathbf{a}_{793} = (0.125, 0, 0, 2)$ corresponds to the RFC 793 RTO [17].

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., $\mathbf{a} = (\alpha, \beta)$, while keeping *n* and *k* at their default values of 1 and 4 respectively. In the second reduced estimator, which we call RTO_3, we additionally allow *k* to vary, i.e., $\mathbf{a} = (\alpha, \beta, k)$, while keeping *n* equal to 1.

Figure 4 shows the optimal RTO_4 curve and 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 4 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 4 certainly gets better with the increase in the number of tuning variables. For a given average overwaiting factor $w = 4.12$ (produced by Jacobson's RTO), RTO_2 and RTO_3 both achieve optimality in the same point and offer only a slight improvement in the number of duplicate packets – 11.15% compared to 12.63%. RTO_4, however, offers a more substantial improvement, achieving $d = 7.84\%$ and keeping the same average overwaiting factor $w = 4.12$.

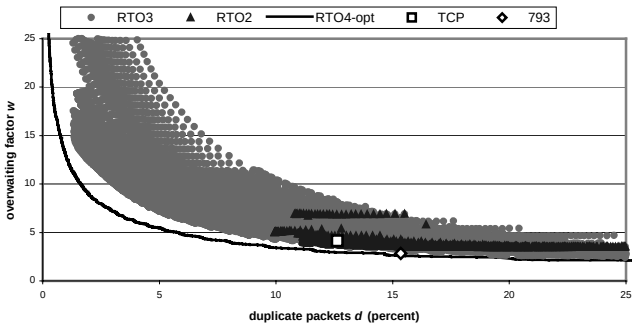

Figure 4. Performance of TCP-like estimators in ISP_a.

Furthermore, Figure 4 shows that the optimal RTO_4 curve (built by the exhaustive search) is convex and fairly continu-

ous until approximately 20% duplicate packets. Consequently, we can build the optimal RTO_4 curve using our reformulated problem with scalar weighted performance function $G(\mathbf{a})$ and compare the results with those in Figure 4. 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 optimizing the weighted performance function $G(\mathbf{a})$ does in fact allow us to produce the optimal RTO_4 curve, we focused on the following minimization problem for a given value of weight M:

$$\min_{\mathbf{a}\in S} G(\mathbf{a}) = \min_{\mathbf{a}\in S}\left(M \cdot d(\mathbf{a}) + w(\mathbf{a})\right) \qquad (11)$$

The fact that function $G(\mathbf{a})$ has unknown (and non-existent) partial derivatives $\partial G(\mathbf{a})/\partial a_k$ 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 [12]) 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 ten times per weight M produced very good results.

Figure 5 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 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. In addition, we should note that for the rest of the paper, we will focus on the performance of RTO estimators within a *reasonable* range of duplicate packets, which we considered to be between 0 and 10 percent.

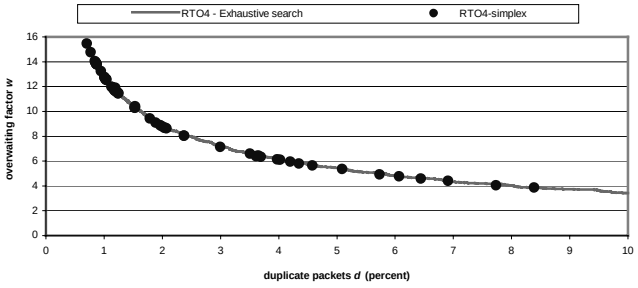

Figure 5. Points built by Downhill Simplex and the exhaustive search in the optimal RTO_4 curve.

Close examination of Figure 5 reveals that both *optimal* curves remind us of a power function in the form of

$$w_{opt} = C(d_{opt})^{-p},\ p > 0. \qquad (12)$$

To investigate this observation further, we replotted the points of the Downhill Simplex curve in Figure 6 on a log-log scale and fitted a straight line to the points. A straight line provides an excellent fit (with correlation 0.9994) and allows us to model the optimal RTO curve as a power function (12) with $C = 1.022$ and $p = 0.55$.

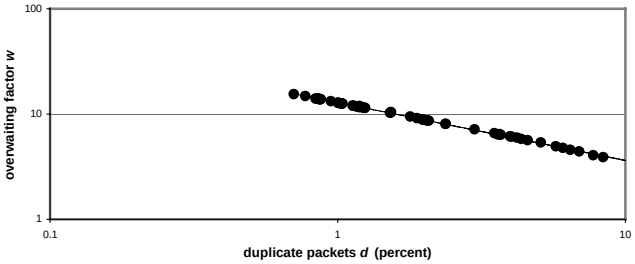

Figure 6. Log-log plot of the optimal (Simplex) RTO_4 curve.

Knowing that the relationship between w and d in the optimal RTO_4 curve is a power function, we can now analytically compute optimal points (d_{opt}, w_{opt}) that minimize function $G(\mathbf{a})$ for a given weight M. Rewriting (10) using the power law in (12), taking the first derivative, and equating it to zero we get:

$$\frac{\partial G(\mathbf{a})}{\partial d_{opt}} = \frac{\partial}{\partial d_{opt}}\left(M d_{opt} + C d_{opt}^{-p}\right) = M - Cp d_{opt}^{-p-1} = 0\ . \qquad (13)$$

Solving (13) for d_{opt} and using power law (12) one more time, we get 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:

$$d_{opt} = \left(\frac{Cp}{M}\right)^{\frac{1}{p+1}} \text{ and } w_{opt} = \frac{M}{p}\left(\frac{Cp}{M}\right)^{\frac{1}{p+1}}\ . \qquad (14)$$

B. Tuning Parameters

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 computed the correlation between RTO_2 estimates and the corresponding future round-trip delays for different values of (α, β). Surprisingly, the highest correlation was achieved at the point where α was 1.0 and β was very close to zero ($\beta = 0.044$). This result gave us an idea that an RTO estimator with (α, β) fixed at (1, 0) should provide a reasonably high correlation with the future RTT, as well as that it was possible to achieve the values of the optimal RTO_4 curve by just varying parameters n and k in RTO_4.

To investigate our hypothesis, we constructed a *reduced* estimator, which we call $RTO_{4(1,0)}$ and which is produced by RTO_4 at input points (1, 0, k, n). We performed an exhaustive search of the reduced tuning domain (i.e., (k, n)) and plotted the range (d, w) of the new reduced estimator in Figure 7 (lightly shaded area). As the figure shows, the optimal simplex RTO_4 curve (shown as squares in Figure 7) 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. The implication of this fact is that it is no longer necessary to maintain a smoothed RTT average to achieve optimality, because $\alpha = 1.0$ means that the SRTT always equals the last RTT sample.

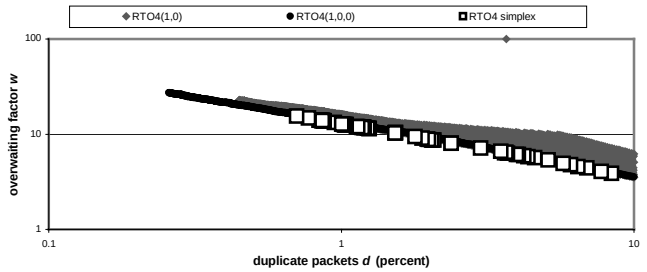

Figure 7. RTO_4-Simplex and two reduced RTO_4 estimators on a log-log scale.

The next logical step was to question the need for SVAR in the RTO_4 estimator (3) since SVAR turns out to be a constant when $\beta = 0$ and most likely can be eliminated without harm. In the same Figure 7, we plot an additional curve, called $RTO_{4(1,0,0)}$, which corresponds to the points produced by RTO_4 at input points (1, 0, 0, n) for different values of 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 the $RTO_{4(1,0)}$ estimator. The optimality of the $RTO_{4(1,0,0)}$ estimator is maintained up to approximately $d = 15\%$, at which time it becomes suboptimal. A straight line fitted to the $RTO_{4(1,0,0)}$ curve in Figure 7 produces a power function (12) with $C = 1.07$ and $p = 0.5456$.

Further investigation discovered that there is a perfectly 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}:

$$n_{opt} = mw_{opt} + b, \tag{15}$$

where $m = 0.86$ and $b = -0.13$. Since we already know the power law between w_{opt} and d_{opt} in (12), we can easily derive the relationship between n_{opt} and d_{opt} in $RTO_{4(1,0,0)}$:

$$n_{opt} = mC(d_{opt})^{-p} + b. \tag{16}$$

Consequently, (16) can be used to adapt the optimal $RTO_{4(1,0,0)}$ estimator to each application's needs (possibly in real-time). For example, if an application specifies that the maximum number of duplicate packets it is willing to tolerate is $d_{opt} = 2\%$, using (12), the optimal overwaiting factor w_{opt} is 9.12 (the corresponding weight M is 248) and using (16), the optimal RTO estimator is given by $RTO_{4(1,0,0)}$ with $n_{opt} = 7.31$.

V. Percentile-based Estimators

In this section, we analyze our second class of RTO estimators. One can argue that the percentile-based RTO estimator, which we call RTO_P, could be a powerful tool in predicting the future value of the round-trip delay based on the distribution density function of the previous RTT samples. Suppose at any time during a session, the history of RTT samples contains s latest samples of the RTT. Consequently, we define a percentile-based RTO estimator to be

$$RTO_P(t) = n\cdot\{RTT_i, \ldots, RTT_{\max(i-s+1,0)}\}_\alpha, \tag{17}$$

where $i = \max i$: $t_i \le t$ and $\{r_1, \ldots, r_n\}_\alpha$ is the α-percentile of set $\{r_1, \ldots, r_n\}$. After conducting a similar Downhill Simplex optimization search of the corresponding tuning domain, we found that although there were three tuning variables in parameter $\mathbf{a} = (\alpha, n, s)$, the optimal performance was always achieved in cases when s was equal to 1 (due to space limitations, we skip a detailed analysis of the optimal RTO_P curve). Since history $\{RTT_i, \ldots, RTT_{\max(i-s+1,0)}\}$ in the optimal case contained only one sample, the value of α did not affect the performance of the optimal RTO_P estimator, which turned out to be equivalent to $RTO_{4(1,0,0)}$. In the previous section, we showed the latter estimator to be almost optimal within the class of TCP-like estimators and derived formulas to convert the amount of duplicate packets d_{opt} to parameter n_{opt}. Our final conclusion on RTO_P is that the extra overhead of maintaining a history of RTT samples is not warranted in the context of RTO estimation in NACK-based applications.

VI. Jitter-Based Estimators

A. Structure and Performance

Our last 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 allows us to reduce the number of duplicate packets compared to RTO_4 by more than 67%.

This is the point when we must discuss a strong conceptual difference between TCP's and NACK-based retransmission schemes. The difference lies in the fact that the distance between RTT samples in real-time applications is large and varies quite randomly during a session (in our experiment, the average distance between consecutive RTT samples was 15.7 seconds), while TCP consistently obtains RTT samples on a per-packet basis.

There are two implications arising from this difference. First, the above analysis of TCP-like RTO estimators should not be taken to literally mean that in actual TCP implementations, the performance of the examined estimators will be similar to what is shown in the figures (in fact, we expect it to be quite different). We can only report the performance of these schemes in NACK-based applications, and further study of the RTT process in real TCP implementations is required before one can reach similar conclusions about TCP's RTO.

Second, 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 probably do not need since TCP senders obtain a substantial amount of RTT samples through their 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 [11], [18] instead of sending one packet every so many milliseconds. Although the latter is considered to be an ideal way of sending video traffic by many researchers (e.g., [4]), in practice, there are limitations that do not allow us to follow this ideal model. Furthermore, bursty packet transmission reduces the overhead of frequent switching between processes, handles varying packet sizes better, and allows more simultaneous streams per server.

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 target bitrate (for comparison, RealAudio servers use D_b = 1800 ms [11]). Each packet in our real-time application carried a burst identifier, which allowed the receiver to distinguish 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 the reason for this was that more cross traffic was able to queue between the bursts of packets 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:

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

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* $SVAR\Delta_j$:

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

and

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

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 the burst duration 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_j is the time when our trace produced the j-th sample of inter-burst delay Δ_j (ideally, T_j equals $t_{first}(j)$) and 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:

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

where $i = \max i$: $t_i \leq t$ and $j = \max j$: $T_j \leq t$. Figure 8 shows the performance of the RTO_J estimator (triangles) along the optimal RTO curve 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 (squares in the figure) in terms of duplicate packets.

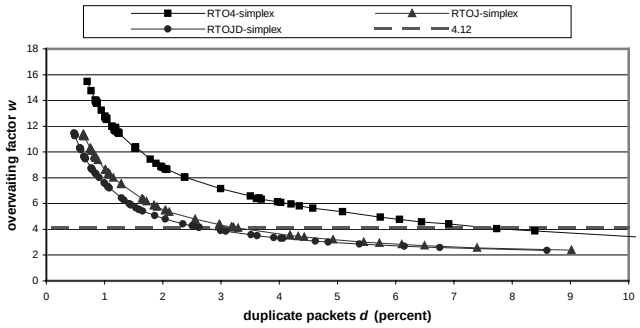

Figure 8. Jitter-based RTO estimators compared with the RTO_4 estimator.

Furthermore, we should mention that an even better estimator, which we call RTO_{JD}, can be created by incorporating the duration between the time of the last RTT sample (i.e., t_i) and the time at which the RTO is being estimated (i.e., t) into the RTO_J estimator:

$$RTO_{JD}(t) = (n + k\cdot(t - t_i))\cdot RTT_i + m \cdot SVAR\Delta_j, \quad (22)$$

where $i = \max i$: $t_i \leq t$, $j = \max j$: $T_j \leq t$, and time units for t and t_i are seconds. Within a reasonable range of d (below 10 percent), RTO_{JD} reduces the number of duplicate packets in RTO_4 by up to 67% as shown in Figure 8 (circles). 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%. Remarkably, RTO_J and RTO_{JD} are now able to improve this value to 3.25% and 2.64% respectively.

B. Tuning Parameters

The RTO_J estimator 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 α_1. 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 allows us to compute $SVAR\Delta_j$ using integer arithmetics. Just as in the RTO_4 estimator, (α_1, β_1) can be fixed at their optimal values and the optimal RTO_J curve is entirely built using n and m.

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

and m_{opt} in the optimal RTO_J curve shown in Figure 8. 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

$$m_{opt} = 4.2792 \cdot n_{opt} - 2.6646 \quad (23)$$

describes the optimal parameters n and m reasonably well. We observed a similar linear trend in RTO_{JD} with slightly different parameters. Furthermore, we noticed that many optimal points in RTO_{JD} were reached when k was equal to 0.5. Consequently, we created two reduced estimators by always keeping m as a function of n shown in (23) and k equal to 0.5. We called the two reduced estimators RTO_{J427} and RTO_{JD427} respectively (427 comes from the constant in front of n_{opt} in (23)) and compared their performance (by running n through a range of values) to that of RTO_J and RTO_{JD} in Figure 9. As the figure shows, both reduced estimators reach the corresponding optimal RTO curves with high accuracy.

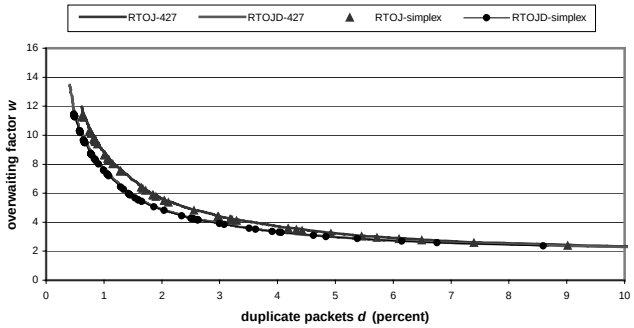

Figure 9. Both reduced jitter-based estimators compared with optimal RTO_J and RTO_{JD} estimators.

Similar power laws (12) and (16) hold for the optimal RTO_J and RTO_{JD} curves, as well as for the optimal RTO_{J427} and RTO_{JD427} curves. TABLE I summarizes the values of constants in both power laws.

TABLE I
SUMMARY OF CONSTANTS IN VARIOUS POWER LAWS

Part I. Power law for optimal RTO curves: $w_{opt} = C(d_{opt})^{-p}$.			
RTO estimator	C	p	**correlation ρ**
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_{JD}	0.51	0.5815	0.9992
RTO_{J427}	0.53	0.6098	0.9991
RTO_{JD427}	0.51	0.5878	0.9980
Part II. Power law for optimal parameter n: $n_{opt} = C_1(d_{opt})^{-p} + C_2$.			
Reduced RTO 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
RTO_{JD427}	0.20	0.27	0.5878

Using the same example from section IV, for $d_{opt} = 2\%$, we find that w_{opt} is 5.75 in RTO_{J427} and 5.07 in RTO_{JD427} (compared to 9.12 in RTO_4). Given parameters in the second half of TABLE I, the values of n_{opt} in the reduced jitter-based estimators are 2.47 and 2.32 respectively (compared to 7.31 in $RTO_{4(1,0,0)}$), and the values of m_{opt} using (23) are 7.91 and 7.30 respectively. As we can see, the superior performance of the 427 estimators 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.

VII. CONCLUSION

Our study of several classes of RTO estimators in NACK-based applications indicate the following:

- the default TCP estimator is a poor choice for NACK-based applications and it can be substantially improved;
- the latest RTT sample has the most relevance to the value of the future round-trip delay due to the large spacing between RTT samples in NACK-based applications;
- frequent delay jitter samples prove to be very helpful in fine-tuning NACK-based RTO estimation and can be used as a good predictor of the *changes* in the future RTTs;
- the average overwaiting time in many studied *optimal* RTO estimators is approximately proportional to the inverse of the square root of the percentage of duplicate packets.

REFERENCES

[1] M. Allman and V. Paxson, "On estimating end-to-end network parameters," *ACM SIGCOMM*, September 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*, September 2000.

[5] 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.

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

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

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

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

[10] D. Loguinov and H. Radha, "End-to-End Internet Path Dynamics for Real-time Streaming Applications," *Under submission*, November 2000.

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

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

[13] J. Padhye, J. Kurose, D. Towsley, and R. Koodli, "A model based TCP-friendly rate control protocol," *IEEE NOSSDAV*, 1999.

[14] C. Papadopoulos and G.M. Parulkar, "Retransmission-based error control for continuous media applications," *IEEE NOSSDAV*, 1996.

[15] V. Paxson, "Measurements and analysis of end-to-end internet dynamics," *Ph.D. dissertation*, Computer Science Department, University of California at Berkeley, 1997.

[16] V. Paxson and M. Allman, "Computing TCP's retransmission timer," *IETF RFC 2988*, ftp://ftp.isi.edu/in-notes/rfc2988.txt, November 2000.

[17] J. Postel, "Transmission control protocol – DARPA Internet program protocol specification," *IETF RFC 793*, September 1981.

[18] H. Radha, Y. Chen, K. Parthasarathy, and R. Cohen, "Scalable internet video using MPEG-4," *Signal Processing: Image Communication*, 1999.

[19] I. Rhee, "Error control techniques for interactive low bitrate video transmission over the Internet," *ACM SIGCOMM*, September 1998.