Jitter Based Delay Boundary Prediction of Wide-Area Networks

Qiong Li* David L. Mills†
Philips Research University of Delaware

May 25, 2001

Abstract

The delay boundary prediction algorithms currently implemented by transport protocols are lowpass filters based on autoregressive and moving average (ARMA) models. However, recent studies have revealed a fractal-like structure of delay sequences, which may not be well suited to ARMA models. In this paper we propose a novel delay boundary prediction algorithm based on a deviation-lag function (DLF) to characterize end-to-end delay variations. Compared to conventional algorithms derived from ARMA models, the new algorithm can adapt to delay variations more rapidly and share delay's robust high-order statistical information (jitter deviation) among competing connections along a common network path. Preliminary experiments show it outperforms Jacobson's algorithm, which is based on an ARMA model, by significantly reducing the prediction error rate. To show the practical feasibility of the DLF algorithm, we also propose a skeleton implementation model.

1 Introduction

In a best-effort packet switching network, packet loss statistics are a good (in fact sometimes the only) means to estimate the network congestion state. Conventional transport protocols detect packet loss by a retransmission timeout (RTO); if after the RTO an ACK is not received, the packet is retransmitted. Unfortunately, if the RTO value is too small, the network is burdened by an unnecessary packet; if too large, the application may stall. In a window-based transport protocol such as TCP, a retransmission event will cause the window to shut down, resulting in a slow-start event and requiring several round trips before steady state is achieved. Clearly, in order to maintain high throughput, TCP needs to set its RTO properly to reduce the probability of false alarms and speed up loss detection.

As network delays change with time, the RTO must adjust as well. In TCP the adjustment is based on a real-time delay boundary prediction using roundtrip delay samples (RTT) and a lowpass filter to predict the smoothed roundtrip delay (SRTT), and another lowpass filter to predict the mean deviation (RTTVAR) between the SRTT and the measured RTT's. The delay boundary (or RTO) is then calculated as the sum of SRTT and RTTVAR multiplied by a constant factor $k = 4$. In addion to the boundary prediction, TCP uses a crafted strategy to select valid RTT measurements and to age RTO within a [MIN, MAX] range. Detailed analysis of the TCP strategy can be found in [15, 10, 9, 1].

The motivation to use a lowpass filter to predict the SRTT, as explained by Jacobson in [9], is that the SRTT is expected to be a mean RTT (or smoothed RTT), which can follow the changing delay trends common in the Internet. However, the physical interpretation of the calculated SRTT has not been clearly identified so far in the literature, based on the authors' knowledge. We think it may be necessary to further clarify what the SRTT really is. As we argue in the following Section 2, the traditional lowpass filter used to calculate the SRTT is a *least mean-square error* predictor of the RTT by modeling the RTT process as an ARMA process, or more precisely the sum of a white noise sequence and a weighted Brownian motion sequence. The physical meaning of the SRTT is then that: if we model all the past RTT samples as a realization of the assumed random process, then the one-step best prediction of the next RTT is the SRTT.

Recent studies [4, 11] show that packet delay processes of wide-area networks have a complex structure: a short piece of a delay sequence densely sampled over a small time scale should be modeled as a nonstationary segment, while a long piece sparsely sampled over a large time scale should be modeled as a long-range dependent (LRD) process. This means a sequence of measured packet delays with an arbitrary sampling period may be modeled as a stationary time sequence, a nonstationary one or a complex combination of both. We call this case a *multi-structure*, which may not be well fitted by an ARMA model (see Section 3 for detail explanation). Empirical studies show that the multi-structure model better fits the observed data [11].

We believe the delay variations in such cases are dominated by queueing delays in the switches as the packet travels across the network. Due to the mathematical intractability of queueing analysis and traffic modeling, it is very hard to directly derive an analytic model to accurately describe the characteristics of queueing delays. For the purpose of calculating RTO, it may not be necessary either. However, there are studies [8, 16] on the jitter of tagged streams in ATM networks in which the probability density functions and correlation functions are derived in explicit forms. In these studies, accurate statistical knowledge about the background traffic and the tagged streams were assumed, which may not be available in the rowdy Internet.

The main contribution of this paper is a heuristic model based on the deviation-lag function (DLF) to characterize packet end-

*Video Communications, Philips Research, 345 Scarborough Road, Briarcliff Manor, NY 10510, e-mail: qxl@philabs.research.philips.com.

†Department of Electrical and Computer Engineering, University of Delaware, Newark, DE 19716, e-mail: mills@udel.edu. This work is supported by DARPA Information Technology Office Contract DABT 63-95-C-0046.

to-end delays (or queuing delays) and produce the delay boundary predictions. Since the delays show different characteristics (nonstationary vs. stationary) in different time scales, the advantage of using the DLF is that it can capture the statistical features in both small time scales and large time scales. Also, a reliable transport protocol may be only concerned about the delays experienced by a tagged packet stream of its packets when statistically multiplexed with other background traffic. Usually the tagged stream is considered only a small contribution to the background traffic and so does not affect the queueing delays. Under this assumption, the DLF can provide a robust characterization of network delays and connections across a common network path may share the same DLF.

The rest of the paper is organized as follows. Section 2 explains the presumption about the SRTT estimator, and discusses how to choose the parameters for the estimator to best fit different conditions. In Section 3, we give a further reasoning on why the delay sequences demonstrate different random structures in different time scales. We then define and discuss the DLF itself and present the delay boundary prediction algorithm. In Section 4, we use both periodic and Poisson distributed ping traces to test the proposed algorithm and compare with Jacobson's algorithm. We show the proposed algorithm outperforms Jacobson's algorithm in terms of lower prediction error rate. In Section 5 we discuss the implementation issues and present a skeleton implementation using a scalable and robust algorithm to maintain the required tables. Section 6 summarizes and concludes this paper.

2 Analysis of the SRTT Algorithm

The SRTT calculation widely implemented in TCP and many applications to predict both the delay boundary and available bandwidth is a lowpass filter. By further study of the delay model assumed by the calculation, we may discover limitations in its use and identify the conditions under which the SRTT is suitable or not. At the same time, the study will motivate why we propose a new algorithm.

In order to explore the model implicit in the SRTT calculation, let us assume that the delays can be modeled by

$$R_n = M + H(q)e_n \tag{1}$$

where R_n is the nth delay sample, M is a constant representing the mean RTT, $\{e_i, i = -\infty, \ldots, n-1, n\}$ is a sequence of independent, identically distributed (*iid*) random variables with zero mean and a certain variance. For convenience we introduce the shift operator q in (1) by $qe_n = e_{n+1}$ and $q^{-1}e_n = e_{n-1}$. $H(q)$ is a filter to be identified in the analysis. If applying system identification theory [13] to this case, we can derive the nth *least mean-square error* prediction based on all the past measurements $\{R_i, i \leq n\}$, that is

$$\hat{R}_n = H^{-1}(q)M + [1 - H^{-1}(q)]R_n \tag{2}$$

where $H^{-1}(q) = 1/H(q)$.

On the other hand, the SRTT is calculated using a low-pass filter

$$\hat{R}_n = (1-g)\hat{R}_{n-1} + gR_{n-1} \tag{3}$$

where $\hat{R}_n$ is the nth of SRTT estimates, R_{n-1} is the $n-1$st delay measurement and g is a constant with range $(0, 1)$. (3) can be rewritten as

$$\hat{R}_n = \frac{gq^{-1}}{1-(1-g)q^{-1}}R_n \tag{4}$$

Assume (3) is the nth optimal estimate of SRTT. Comparing (2) and (4) and letting $H^{-1}(q)M = 0$, we immediately discover that

$$H^{-1}(q) = \frac{1-q^{-1}}{1-(1-g)q^{-1}} \tag{5}$$

which suggests that

$$\begin{aligned} H(q) &= \frac{1-(1-g)q^{-1}}{1-q^{-1}} \\ &= 1 + g\sum_{k=1}^{\infty} q^{-k} \end{aligned} \tag{6}$$

$H(q)$ is the filter we want to identify in (1), which means that if we take (3) as the best estimator of SRTT in the sense of *least mean-square error*, we have assumed that the delays can be modeled by (1) with the filter (6).

We substitute $H(q)$ in (1) by (6), so

$$R_n = R_{n-1} + e_n - (1-g)e_{n-1} \tag{7}$$

or

$$R_n = M + e_n + g\sum_{k=1}^{\infty} e_{n-k} \tag{8}$$

which (7) indicates an ARMA model. (8) reveals more interesting features about the model. From (8) we see that when $g \to 0$, the delays is modeled by a white noise process, but when $g \to 1$, a sequence of Brownian motions, which is a nonstationary process. When $0 < g < 1$, the delays are modeled as the *sum of a white noise process and a weighted nonstationary process*. It is clear that SRTT is not the estimation of the mean RTT (or M) in the model (1), but the best real-time estimator of RTT based on the model (8). Since $\{e_i, i = -\infty, \ldots, n-1, n\}$ with zero mean, the unbiased estimator of M should be

$$\hat{M} = \frac{1}{n}\sum_{k=1}^{n} R_k \tag{9}$$

(8) reveals the limitation of the SRTT estimate (3). With a fixed g, as implemented in TCP, this method presumes that all measured delay sequences can be uniformly modeled by (8), independent of sampling method (regular or irregular), sampling frequency (high or low), and the underling path characteristics. This is certainly not true for the reasons elaborated in the next section.

The low computing overhead of the current SRTT calculation makes it attractive for typical applications. (8) provides insight in how to choose g for (3) under different protocol requirements and network conditions. As further explained in the next section, delay sequences densely sampled in small time scales are more nonstationary in nature than stationary because of the unavoidable queuing phenomenon in packet switches, while sparsely collected sequences in large time scales should be considered stationary or quasi-stationary after canceling the diurnal effect. Therefore, to match the model (8) closer to different real situations and to obtain better performances of (3), we should choose a large g when calculating the SRTT for packets clustered together (a burst of packets), but a small g for a string of packets sparsely separated from each other.

The question of which time scale best expresses the delay characteristics in blending from nonstationary to stationary has been studied in [12]. For the case of TCP, if we estimate the SRTT only once per roundtrip time, corresponding to a sparsely sampling case, a small g is desired; but if for every segment in flight, we should choose a large g, so that the SRTT catches up to the nonstationary process more quickly.

The above analysis suggests that the existing SRTT calculation method may not be optimal for all situations, at least if using a fixed set of parameters. However, designing a parameter adaptation scheme for the lowpass filter (3) which is optimal under different conditions is also a nontrivial task. This observation motivated us to explore new ways to characterize delays and develop new delay boundary prediction algorithms.

3 Prediction Algorithm

It is necessary to explore the nature of delay variations and the correlation between measured delays in order to understand the logic of our delay boundary prediction algorithm. In this section we explain why delay sequences are multi-structured. Then we introduce a powerful delay characterization tool the DLF. Based on the DLF analysis we further introduce the concepts of first-order and second-order delay dependencies. The existence of the two types of naturally different dependencies illustrates the limitation of the ARMA models we discussed previously in Section 2. Finally in this section, we present the proposed delay boundary prediction algorithm.

3.1 Characteristics of Delays

Based on Markov models, conventional queueing theory predicts that, if inter-arrival time and service time are both independent, the queueing delays can be modeled as a regeneration process [5]. A regeneration process includes many statistically similar regeneration cycles, each of them corresponding to a queueing busy cycle (see Figure 1 for the illustration of busy cycle). Within each cycle, the queueing delay behaves like a random-walk, which is a nonstationary process. Therefore, a queueing delay process is nonstationary in time scales comparable to the average regeneration cycle, or the average queue busy period, but stationary in time scales much larger than that.

Because of the closed-loop feedback flow control mechanisms used by transport protocols and the bursty characteristics of application data, the arrivals are no longer independent and Markov models may not be appropriate. A general queueing delay process is no longer a pure regenerative process in the strict sense. However, as time progresses we still see the queueing system either in a busy period or in an idle period if the utilization factor $\rho < 1$. If the queue buffer is large enough, within each busy period the fluctuation of the queue length is still a random-walk-like process, and should be modeled as a tiny nonstationary process (or segment). Therefore, a queueing delay process in general still maintains the same random structure as a regenerative process — stationary as a whole but constituting an infinite number of nonstationary busy periods. Due to the bursty property of the incoming traffic, the average length of a nonstationary busy period tends to be longer than that predicted by Markov models under the same average arrival rate. More precisely, since real traffic is long-range dependent, the behavior of delays in small time scales corresponding to these busy periods changes from a random-walk-like to a more fractal-like [11].

In general, we may say that for a stable queueing system, the queueing delays can be modeled by a *stationary process with nonstationary sub-processes*. Such processes have what we call a *multi-structure*, whose meaning will become clearer after we introduce the jitter deviation-lag function (DLF) analysis in Section 3.2. Furthermore, we will show that the multi-structure model can also be applied to the case of end-to-end paths consisting of multiple hops and used to predict end-to-end delays.

A transport protocol sees only the queueing delays of a tagged packet stream in which its traffic is embedded. As a tagged packet travels over a network path, it can in principle sample the delays of all the queues along the path. If the intervals between the interdeparture times of tagged packets are small compared to the average busy periods of the queues, individual packets in the stream will, with high probability, be able to take more than one sample from the same busy period, as shown in Figure 1.(a). This implies that the measured delay samples from the stream will display a multi-structure characteristic. On the other hand, if the intervals between tagged packets are much larger than the average busy period, then the tagged stream will sample many different busy periods, as shown in Figure 1.(b). In this case the delay measurements fail to reveal the nonstationary structure of the underlying busy period.

Figure 1 explains why the characteristic of a densely sampled delay sequence is dramatically different from that of a sparsely sampled one. An ARMA model with a set of parameters which match one extreme will not be able to match the other. This is why we began to explore the multi-structure model in the first place.

3.2 Jitter Deviation-Lag Function Analysis

Since a densely sampled delay sequence may capture the multi-structure characteristic, we need an analysis tool which can clearly identify and quantify this characteristic. Conventional queueing theory does not provide means to characterize queues in a way that can display these features. Conventional autocorrelation function analysis is useful for single-structure stationary processes, but not multi-structure processes with nonstationary cycles. Therefore in the following, we introduce an analysis tool called the jitter deviation-lag function (DLF), which can characterize delay variations in different time scales.

In order to show why we chose the jitter DLF analysis method, we further explore what kind of nonstationary nature a queueing delay process may have in small time scales. Define

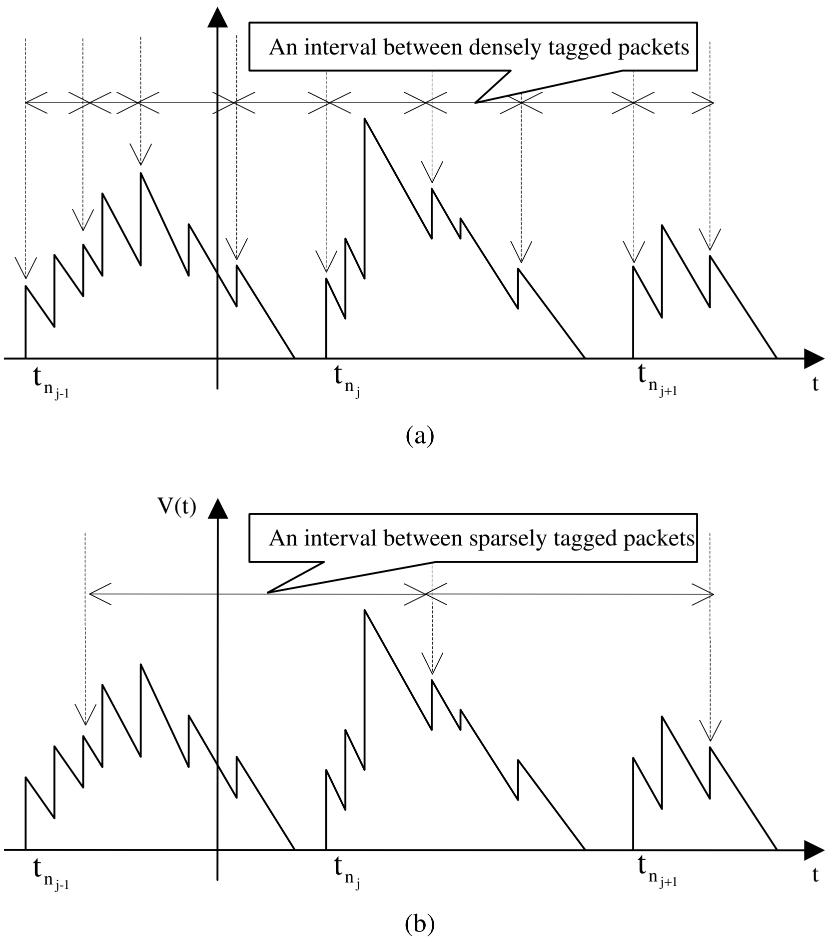


Figure 1: A densely sampling case of $V(t)$ (a) and a sparsely sampling case of $V(t)$ (b). $V(t)$ is an instance of a queueing delay process.

$a_k = s_k - \sigma_{k+1}$, $k = 1, 2, \ldots$, where s_k is the service time of the kth arrival and σ_{k+1} the interval between the kth and the $k+1$th arrivals. The queueing delay $V(t)$ in a busy period $t \in [t_{n_j}, t_{n_j} + b_j]$, where t_{n_j} is the start time of the jth busy period and b_j its duration, can be calculated by

$$V(t) = \sum_{k=n_j}^{n} a_k - (t - t_n), \tag{10}$$

where $t_n < t < t_{n+1}$, and t_n is the nth arrival time. (10) demonstrates that $V(t)$ in a busy period can be calculated by the sum (or integral) of a subsequence of a_k. If a_k is a white-noise sequence, then mathematically $V(t)$ will behave as a random walk (or Brownian motion) process, which is nonstationary. If a_k is a long-range dependent sequence, as is more likely the case in reality, $V(t)$ will behave like a fractional Brownian motion (fBm) process (see [11] for a detailed discussion). Both random walk and fBm are nonstationary fractal phenomena [6][1]. Therefore, a queueing delay process inherently has a fractal structure in time scales smaller than the busy periods, regardless of the arrival process, as long as the traffic can persist for some time in the buffer queue.

A general way to characterize a nonstationary fractal process is to conduct a scaling analysis, namely, to find out in what range the process fluctuates when looking at the process in different time-scales. This is similar to zooming in and out while looking at the delay time series graph. A direct approach to characterizing fractal processes is by defining the *height difference correlation function* [14] as

$$C_2(\tau) = \{\mathbf{E}[(\delta X(\tau))^2]\}^{\frac{1}{2}} = \{\mathbf{E}[(X(t+\tau) - X(t))^2]\}^{\frac{1}{2}} \tag{11}$$

where $X(t)$ is a fractal curve (or signal). $C_2(\tau)$ calculates the deviation of the increments of $X(t)$ in time scale τ. For a single-structure, nonstationary fractal signal, $C_2(\tau)$ will have the form

$$C_2(\tau) \sim \tau^H \tag{12}$$

where $0 < H < 1$ is the Hurst parameter of the signal. (12) shows that $X(t)$ has a power-law scaling behavior in all time scales (or all correlation lengths). Thus the $\log C_2(\tau)$ vs. $\log(\tau)$ plot will be linear over the scaling domain $\tau \in [0, +\infty)$ with the slope of the plot equal to H. For example, an ideal fBm

[1]By definition, fBm is the integral of a stationary long-range dependent process with Gaussian distribution, such as fractal Gaussian noise (fGn), while Bm is simply the integral of white Gaussian noise. Both fBm and Bm are non-stationary and self-similar. Bm is a special case of fBm with Hurst Parameter $H = 0.5$.

$X(t)$ will produce such a linear plot with the slope equal to its H over the entire scaling domain. However, If $X(t)$ is a multi-structure signal, such as, it behaves like fBm in small time scales (or small τ's) while stationary LRD noise in large time scales (see Section 3.3 for further explanation on the concept of multi-structure), then due to the multi-structure characteristic, (11) may not show a uniform power-law scaling behavior as (12). The linear pattern of the $\log C_2(\tau)$ vs. $\log(\tau)$ plot may break at some scale points (τ's).

To see why the plot may depart from straight line, we expand (11) as the following:

$$C_2(\tau) = \{\mathbf{E}[X(t+\tau)^2] + \mathbf{E}[X(t)^2] - 2\mathbf{E}[X(t+\tau)X(t)]\}^{\frac{1}{2}}$$

When τ is small, $C_2(\tau)$ behaves like fBm, which is monotonically increasing with τ. However, when τ exceeds a critical value, say $\tau > \tau_B$, a transition point occurs. At this point the nature of $X(t)$ changes from nonstationary (fBm) to stationary (LRD) and $\mathbf{E}[X(t+\tau)^2] = \mathbf{E}[X(t)^2]$. Furthermore, because at this stage τ has a relatively large lag, the dependency between $X(t+\tau)$ and $X(t)$ is very weak. Normally we have

$$\mathbf{E}[X(t+\tau)^2] + \mathbf{E}[X(t)^2] \gg 2\mathbf{E}[X(t+\tau)X(t)]$$

so $C_2(\tau)$ is almost constant for $\tau > \tau_B$. See Section 3.3 for an example of $C_2(\tau)$.

Therefore, by calculating the height difference correlation function we can determine whether a process has a multi-structure characteristic or not based on the pattern of the function. We apply the height difference correlation analysis to the delay sequences considered previously. Let $X(t)$ be the delay of a packet departing at time t, then $C_2(\tau)$ represents the standard deviation of the delay differences (jitter) of tagged packets separated by a lag τ. Note the mean of jitter is zero for a stable system. We will call this function the jitter *deviation-lag function* (DLF).

3.3 The Typical Pattern of DLF

Empirically, we know a typical jitter DLF of a single queue plotted in log-log scales may look like the curve shown in Figure 2.(a), which shows a multi-structure process. The DLF increases with τ monotonically when τ is small. When τ reaches a critical scale point (τ_B), the DLF assumes a constant value afterward ($\tau > \tau_B$). The two distinguishable sections of the DLF before and after τ_B reflect two distinguishable scaling behaviors — nonstationary and stationary. The first slope section of the DLF can be fitted to a power-law implying a fractal-like (or self-similar-like) behavior with the Hurst parameter H equal to the slope of this section. The second flat section (the constant section) suggests a stationary, noise-like behavior. The transition area around point (τ_B, $C_2(\tau_B)$) can be clearly identified on the DLF. τ_B is referred as the *crossover* scale-point of a delay sequence from nonstationary to stationary. It means two things:

- If we down-sample the delay sequence with a down-sampling period longer than τ_B, the extracted subsequence will appear as a stationary process.
- A small window of delay samples randomly snipped from the original sequence with the window width shorter than τ_B will appear as a nonstationary process.

The transition point (τ_B, $C_2(\tau_B)$) and the shape of the transition area (a smooth transition against a sharp one) of the DLF is determined by the traffic characteristics. We would expect that τ_B equals the average queue busy period. Normally, under light load conditions, the transition is smooth; while under heavy load conditions, the transition is a sharp breakpoint of the DLF. Figure 2.(b) and (c) show the DLF for two ping traces. One trace was produced while sounding the Internet using ping packets at 20ms intervals. The other was produced using the NS simulator and a synthetic network. Both DLF's show patterns similar to Figure 2.(a), suggesting the multi-structure model indeed captures the behavior of typical network paths.

The nature of delay sequences measured over the span from small time scales to large time scales depends on the background traffic. If the traffic is memoryless, then a sequence will behave as Brownian motion in small time scales and white noise in large time scales. However, if the traffic is long range dependent (LRD), the sequence will behave as fBm in small time scales and LRD noise in large time scales. See [11] for a detail discussion on this issue.

3.4 The First-Order and Second-Order Dependencies

In order to better understand how τ_B correlates with queue busy periods, we introduce the concepts of first-order and second-order dependencies. If two delay samples are taken from the same busy periods, there will be a strong correlation between the two samples due to the busy period's nonstationary nature. We call this type of correlation the *first-order dependency*. If samples are taken from different busy periods, the correlation will be relatively weak. We call this type of correlation the *second-order dependency*. The first-order dependency is invoked by the queueing process or the buffer effect. But what causes the second-order dependency? To show the nature of the second-order dependency, we analyze a sampling scenario.

Assume $V_{n_j}(t_1)$ and $V_{n_{j+1}}(t_2)$ are delay samples taken from different busy periods starting at time t_{n_j} and $t_{n_{j+1}}$ respectively. It is easy to see that $V_{n_j}(t_1)$and $V_{n_{j+1}}(t_2)$ can be completely determined by the arrivals in the time period $[t_{n_j}, t_1]$ and $[t_{n_{j+1}}, t_2]$ separately, because all busy periods begin with an empty queue state. If there is no correlation between traffic in the two periods, such would be the case with Poisson traffic, then the two samples are independent. However, if the traffic is LRD, then the correlation between the traffic in the two busy periods could be significant and some dependency could exist between them, which further invoke the dependency between the delays. This is where the second-order dependency comes from. Compared to the first-order dependency, the second-order dependency is quantitatively very weak. The transition from the first-order to the second-order dependency is the reason why the DLF pattern changes from slope-fit to flat-fit around τ_B.

Since the first-order and the second-order dependencies are caused by different mechanisms, so they are naturally different

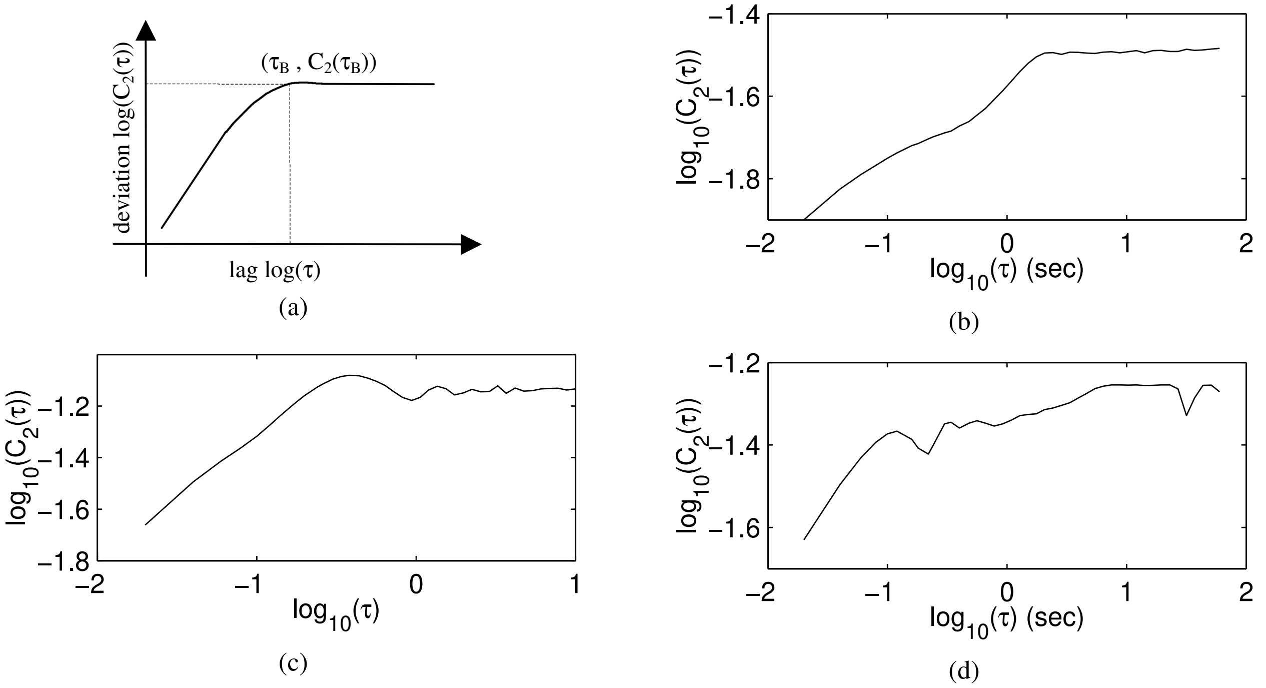


Figure 2: (a) A typical pattern of jitter DLF, (b) the DLF of an Internet ping trace, (c) the DLF of a NS ping trace, (d) the DLF of an Internet ping trace with more than two sections.

from each other. A densely sampled delay sequence may have both types of dependencies, while a sparsely sampled one may have only the second-order dependency. With an LRD background traffic, a sparsely sampled delay sequence will also be LRD. The two types of dependency result in the multi-structure characteristic illustrated previously. Having a clear picture of this characteristic can help to make better use of delay measurements in protocol designs, application implementations and network state monitoring and diagnosis.

The first-order and the second-order dependencies also reveal why ARMA delay models are inadequate to explain the multi-structure characteristic. It is not possible to model the two dependencies with a single ARMA model, especially because of the abrupt transition between the two types of dependency. DLF analysis, on the other hand, can show the transition in a clear way, so it is an efficient tool to characterize multi-structure phenomenon.

3.5 The Relation between Path DLF and Queue DLF

A typical Internet path consist of multiple tandem queues along the path. Under conditions where the Kleinrock independence assumption is valid [3], then the DLF of the path is simply the sum of the DLF for all queues; however, the sum may have more than two distinguishable sections. The first section reflects the behavior at the bottleneck link, while the remaining sections show the behavior at other links. For example, the DLF in Figure 2.(d), which has three distinguishable sections, is calculated from a ping trace of an Internet path. Due to the complexity of Internet traffic and topology, each distinguishable section may not necessarily be strictly linear, suggesting a more complex scaling behavior. But no matter what traffic is involved, the path DLF quantitatively characterizes how the delays vary in different time-scales. Furthermore, as long as the Internet traffic is stationary or quasi-stationary over a period of time, the path DLF will be robust, no matter what it may look like. In fact, the shape may serve as a distinguishing fingerprint of the path over some period of time.

3.6 DLF Based Prediction Algorithm

Under the assumption that network traffic is stationary over some period of time, the delay jitter characteristics of a tagged stream with given lag will have a stationary probability distribution. The standard deviation of the distribution is one point on the path DLF curve. The DLF curve summarizes the standard deviations of all the stationary probability distributions as the lag varies over the measured range. According to the Chebychev inequality

$$P[|\delta X(\tau)| > \delta X_B] \leq \frac{C_2^2(\tau)}{\delta X_B^2} \tag{13}$$

where $\delta X(\tau)$ is the delay jitter of two packets with a lag τ between them, $C_2^2(\tau)$ is the variance of the delay jitter distribution, δX_B is an assumed boundary. Note $\mathbf{E}[\delta X(\tau)]$ is zero. The Chebychev inequality means that if we take $\delta X_B = 10C_2(\tau)$ as an example, then 99% of the $\delta X(\tau)$ will fall into the range $[-\delta X_B, \delta X_B]$, no matter what the stationary jitter probability distribution may be. δX_B provides a jitter boundary estimate with 1% error probability.

Assume $|\delta X(\tau)| = |X(t_2) - X(t_1)|$, and take $\delta X_B =$

$kC_2(t_2 - t_1)$. Invoking (13), we get

$$P[X(t_1) - kC_2(t_2-t_1) < X(t_2) < X(t_1) + kC_2(t_2-t_1)] \geq 1 - \frac{1}{k^2} \quad (14)$$

(14) means that, given the packet delay $X(t_1)$, we can predict a boundary for the packet delay $X(t_2)$ by

$$X(t_1) - kC_2(t_2 - t_1) < X(t_2) < X(t_1) + kC_2(t_2 - t_1) \quad (15)$$

with an error probability $1/k^2$. So the prediction algorithm is simply: *based on an earlier delay $X(t_1)$, and the DLF $C_2(\tau)$ of a path, we can predict the delay boundary of a packet departing at time t_2 ($t_2 > t_1$) by (15) with a prediction error rate determined by the chosen k.* We see that this algorithm is independent of the DLF, as long as the background traffic is stationary.

If the jitter characteristics at different lags can all be fitted to a normal distribution, then a $k = 2.6$ can achieve a prediction error rate less than 1%. In reality the jitter distributions may have heavier tails than a normal distribution, which suggests a larger k (> 2.6) may be necessary to achieve the same prediction error rate. Experiments described in the next section show that $k = 4$ (far less than the 10 indicated by Chebychev inequality) can actually achieve a prediction error rate less than 1%.

An important issue on implementing the algorithm is how to construct and update the path DLF. Section 5 discusses this issue in detail.

4 Performance Comparison

This section describes experiments to test the performance of our proposed DLF algorithm by comparing it with Jacobson's algorithm. The experiments are done using a set of ping traces with either equal interval, or Poisson interval between delay samples.

Jacobson's algorithm has three state variables — SRTT, RTTVAR (RTT variation) and RTO (Retransmission TimeOut) [1, 9]. The SRTT is calculated by (3) with parameter $g = 1/8$; The RTTVAR is updated by a similar formula with a gain $g_1 = 1/4$:

$$RTTVAR_n = (1-g_1)RTTVAR_{n-1} + g_1|SRTT_{n-1} - R_{n-1}| \quad (16)$$

and then the RTO is defined by

$$RTO_n = SRTT_{n-1} + k_j RTTVAR_n. \quad (17)$$

with $k_j = 4$. For this study, we take the RTO_n as the delay boundary prediction of the nth packet calculated by Jacobson's algorithm.

Unlike Jacobson's algorithm, which is driven by a single trace for the prediction, the DLF algorithm involves two traces collected at different times on the same network path. The trace collected at an earlier time is used to calculate the path DLF stored in memory as a table. A DLF table consists of multiple rows but only two columns; the first column stores discrete lags, and the second the corresponding DLF values. This table is then used to predict the delay boundaries for the second trace collected at a later time. The prediction is calculated as

$$RTTB_n = X(t_{n-1}) + k_d C_2(t_n - t_{n-1}) \quad (18)$$

where $RTTB_n$ is the delay boundary prediction of the nth packet, t_{n-1} and t_n are the departure time of the $n-1$st packet and the nth packet, respectively. $C_2(t_n - t_{n-1})$ is taken from the table indexed by $\tau = t_n - t_{n-1}$. We choose $k_d = 3$ in our experiments in order to compare the two algorithms fairly. With these values of k_d and k_j, the two algorithms running over the same trace will produce roughly the same average boundary predictions ($\mathbf{E}[RTTB_n] = \mathbf{E}[RTO_n]$). The goal of the comparison is to see which algorithm can then produce a better prediction of delay changes by determining how well the predicted value bounds the next delay, as measured by the correct-prediction rate (or probability). Of course, both algorithms will never fail if k_j and k_d are chosen large enough, but in that case there can be no basis for a fair comparison.

There are four pairs of ping traces measured from four chosen Internet paths in the experiments. The two traces of each pair were measured from the same path, one with equal interval, the other with Poisson interval. Of each trace pair, the trace with equal interval was collected roughly 1 minute earlier than the trace with Poisson interval. Each one of the eight traces was also divided into two half-traces; each half-trace contains roughly 5000 pings spanning over roughly 100 seconds (corresponding to an average sending rate pkt/20ms). The ping program took a rest of 10 seconds between the collecting of the two half-traces of each trace. We ran both the prediction algorithms on all the second-half-traces, and saved all the first-half-traces for constructing the DLF tables for the DLF algorithm. Table 1 gives a summary of the set of ping traces.

The DLF algorithm described above assumes that, if we try to predict the delay boundary of a packet departed at t_2 based on the measured delay of a packet departed at t_1, we can always obtain a jitter deviation $C_2(\tau_0)$ from the DLF table precisely indexed by a lag $\tau_0 = t_2 - t_1$. But a trace, even with equal interval, can only let us construct a table with discrete lags. The step length of the lags in this case equals the ping interval. A trace with Poisson interval may not even allow us to construct a precise DLF indexed by discrete lags; we believe this may be very likely the case if we would come to a point in the future to implement the DLF algorithm.

We ran our experiments over the traces in three different scenarios:

1. Construct tables with precise DLF values indexed by discrete lags based on traces with equal interval to predict the delay boundaries of traces also with equal interval

2. Construct tables as above, but predict the delay boundaries of traces with Poisson intervals. Since the lags between pings may not be precisely matched by lag entries in the tables. The compromise we make is to use the lag found in the tables which is equal to or greater than the desired value.

3. Construct approximate DLF tables with their lag entries representing small lag domains (the whole lag domain is divided into a number of small lag domains) instead of unique lag values based on traces with Poisson intervals to predict the delay boundaries of traces also with Poisson intervals. In this case the lags between pings may fall into

Table 1: Summary of the ping traces collected from a common source

Trace Name	Interval Type	Min. Delay(ms)	Loss rate %	Destination
uk_e	Equal	96	0.04	UK
uk_p	Poisson	96	0.06	UK
ar_e	Equal	433	0.45	Argentina
ar_p	Poisson	433	0.40	Argentina
au_e	Equal	428	0.04	Australia
au_p	Poisson	428	0.00	Australia
us_e	Equal	76	0.26	USA
us_p	Poisson	76	0.14	USA

one of the small lag domains found in the tables. This scenario is set to simulate a possible implementation of this algorithm.

4.1 Scenario I: Boundary Prediction of Traces with Equal Interval

In this experiment, the DLF tables are built from the first-half-traces of traces with equal interval ($T = 20$ms). As we mentioned before, a DLF table consists of multiple rows but only two columns. The first entry of each row contains the lag iT, where i is the row index of the table. The second entry of the same row contains

$$C_2(iT) = \sqrt{1/(N-i)\sum_{n=1}^{N-i}(X(t_{n+i}) - X(t_n))^2} \qquad (19)$$

where N is the total number of the samples in the first-half-trace. This table is used by the DLF algorithm to predict the delay boundaries of packets in the second-half-trace of the same ping trace. The DLF algorithm is initialized as

$$\begin{aligned} X(t_0) &= \mathbf{E}[X(t_n)] \\ RTTB_1 &= X(t_0) + k_d C_2(\infty) \end{aligned}$$

Where $C_2(\infty)$ is the second entry of the last row in the constructed table.

For comparison we ran both the DLF algorithm and Jacobson's algorithm on the second-half-traces of trace *uk_e*, *ar_e*, *au_e* and *us_e* all with equal intervals. We also repeated the experiment on their downsampled subtraces with different downsampling periods. In this way we explored the performance of the algorithms when applied to traces with different ping intervals.

Figure 3 – 6 shows the experimental results. In all the figures, the x-axis marked as "lag" represents the interval between two pings in a subtrace, so each integer point on the x-axis represents a different subtrace ($lag = 1$ represents the original trace). The y-axis indicates the correct prediction rate. The figures show that, except the experiment on trace *uk_e* and its subtraces, which does not show a clear better performance of DLF algorithm than that of Jacobson's algorithm, the experiments on the other three traces and their subtraces do show a distinctively better performance of the DLF algorithm in that it achieves a higher correct-prediction rate than Jacobson's algorithm, under the condition that both algorithms output approximately the same average boundary predictions (shown in the same figures; the differences are relatively very small compared to the average boundary predictions themselves). In general, there is roughly 3% average difference between the correct prediction rates, which suggest a 60% improvement of the error rate from an average 5% error-rate down to 2% error-rate. In other words, the DLF algorithm avoids 60% of the false predictions made by Jacobson's algorithm. For trace *uk_e*, the experimental result shows the DLF algorithm outperforms Jacobson's algorithm on subtraces with small lags, but not on subtraces with large lags.

4.2 Scenario II: Boundary Prediction of Traces with Poisson Interval

In this scenario the DLF algorithm uses the DLF tables built for Scenario I to predict the delay boundaries of the second-half-traces of trace *uk_p*, *ar_p*, *au_p* and *us_p*, which all use Poisson intervals. Note the tables are built from traces collected earlier than the traces used in the prediction tests. Again, for comparison we ran both the DLF algorithm and Jacobson's algorithm over the same half-traces and their down-sampled subtraces.

Figure 7 - 10 shows the experimental results. This time the x-axis in the figures marked as "average lag" means the average interval between the pings in a subtrace. Each integer point on the x-axis again represents a different subtrace as in Scenario I. The figures clearly show that the DLF algorithm persistently outperforms the Jacobson's algorithm in all four cases roughly by a 3% average gain in the correct prediction rates, suggesting again a 60% improvement of the prediction error rate. Since the DLF algorithm has made compromises when searching through the DLF tables to select a lag slightly larger than the desired one, one may expect that the average delay boundary predictions of a trace and its subtraces calculated by the DLF algorithm are larger than that calculated by Jacobson's algorithm. If it is always the case we can not confidently conclude that the DLF algorithm does a better job, but the figures show that in the cases of trace *uk_p* and *ar_p*, the DLF algorithm produces larger average boundaries, while in the cases of trace *au_p* and *us_p* the Jacobson's algorithm produces larger average boundaries. The absolute difference between the two predictions is still very small compared with the predictions themselves. So we conclude it is justified to compare the performances of the

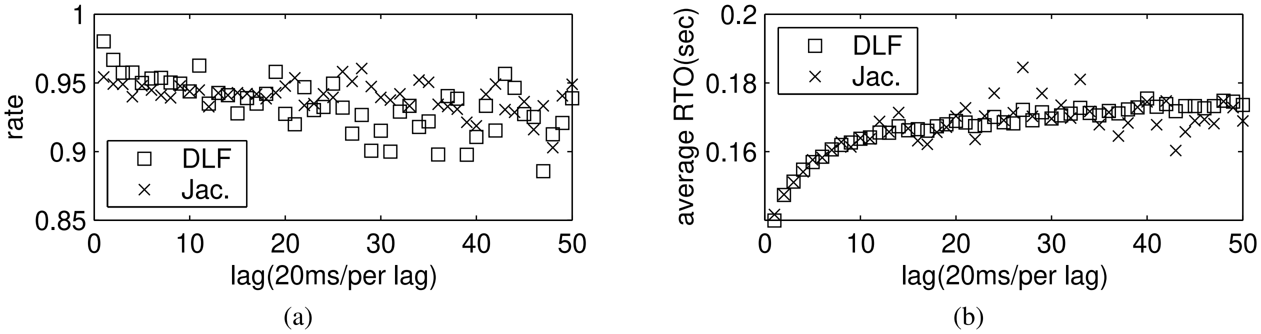


Figure 3: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *uk_e* and its down sampled subtraces.

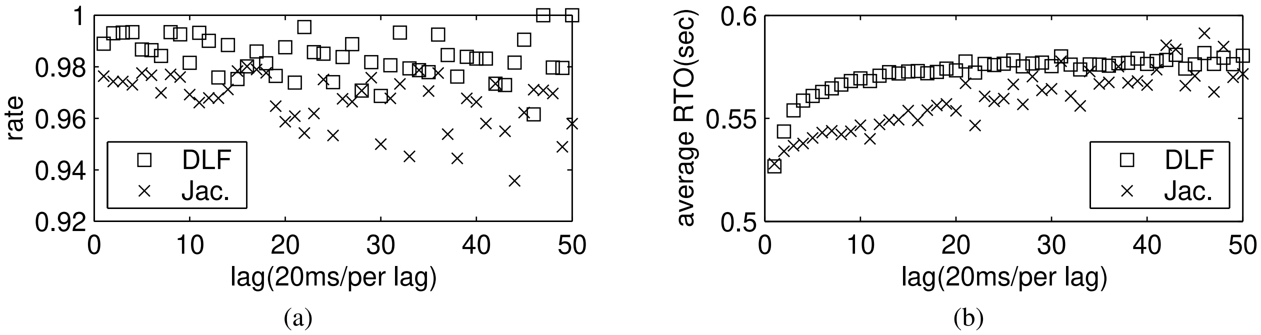


Figure 4: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *ar_e* and its down sampled subtraces.

two algorithms under the settings described earlier.

4.3 Scenario III: Using Approximate DLF Table to Predict Trace Boundaries

This scenario is designed to assess the realizability of a DLF algorithm implementation.

In practice we may not have a periodically sampled delay trace to construct a DLF table. However, it may be easy to collect a trace with randomly distributed intervals between samples from a path. That means we may not be able to build a table with evenly separated discrete lags. Instead, an approximate table from a randomly collected trace may be constructed. In this case the lag entry of each row in the table does not simply represent a single discrete lag iT, but a lag domain (or range) $[\tau_i, \tau_{i+1})$, the second entry in the same row is calculated as

$$C_2(i) = \sqrt{\frac{1}{N}\sum_{n=1}^{L}\sum_{j}(X(t_{n+j}) - X(t_n))^2} \qquad (20)$$

where $j \in \{\tau_i < t_{n+j} - t_n < \tau_{i+1}, 0 < j < L - n\}$, L is the length of the trace, N is the total number of sample pairs, each of which consists of two delay samples with a lag $\tau_i < \tau < \tau_{i+1}$ between them. We see that $C_2(i)$ actually lumps together all the jitter deviations with lags falling into the domain $[\tau_i, \tau_{i+1})$. The DLF algorithm will simply pick up a lumped jitter deviation from the table depending on which domain the lag falls into.

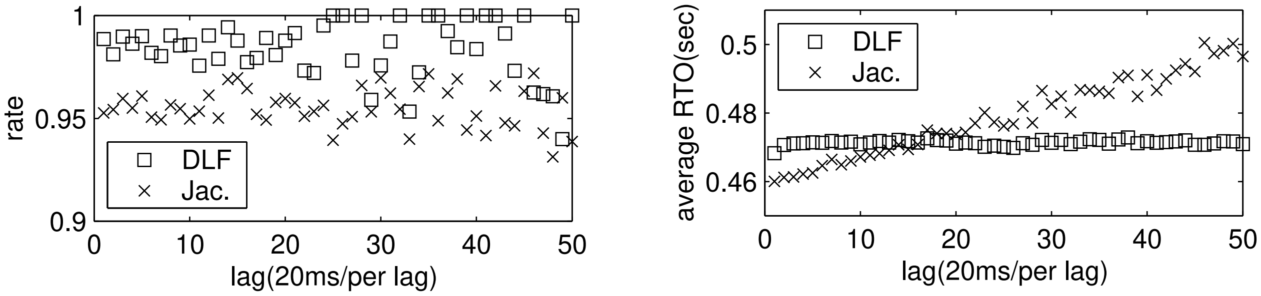
This raises an interesting question: how to divide the lag domain for the table? This question can be formally defined as an optimization problem: Assuming $C_2(\tau)$ is the jitter DLF, $f(\tau)$ is the probability density function (pdf) of lags in a certain situation, and the lag domain has been divided in $M + 1$ small sub-domains; find a set of lags $0 = \tau_0 < \tau_1 < \tau_2 \cdots < \tau_M < \tau_{M+1} \to \infty$, which minimize the mean-square error:

$$W = \min_{\tau_i, i=1,2,\ldots,M} \sum_{i=0}^{M} \int_{\tau_i}^{\tau_{i+1}} (C_2(\tau) - C_2(i))^2 f(\tau) d\tau \qquad (21)$$

where

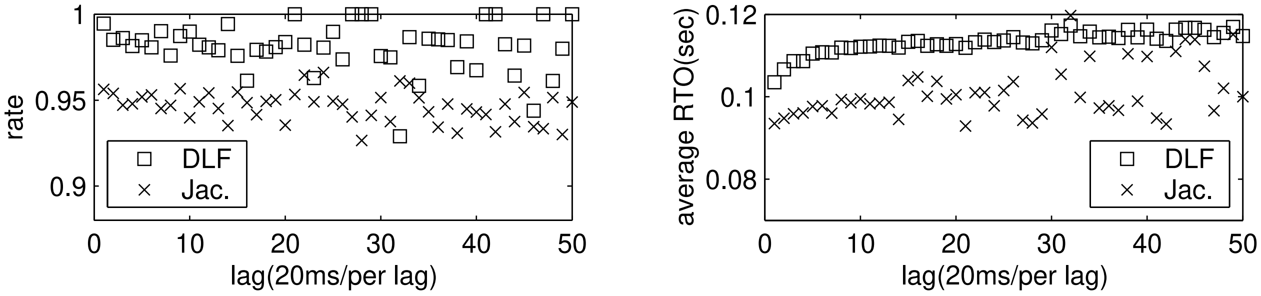
$$C_2(i) = \frac{\int_{\tau_i}^{\tau_{i+1}} C_2(\tau) f(\tau) d\tau}{\int_{\tau_i}^{\tau_{i+1}} f(\tau) d\tau}$$

There is no explicit solution to this problem, so we do not try to find the optimal divisions. Instead, to reduce the division error, we visually check the jitter DLF and the lag pdf, and devise a division scheme that takes into account the generic patterns of both $C_2(\tau)$ and $f(\tau)$. It is obvious that more entries in the table allow the algorithm to produce better predictions, but greater overhead. A certain tradeoff is necessary. The lag domain division scheme we choose in this experiment is: $M = 8$, $\tau_1 = 0.01$sec, $\tau_i = \tau_{i-1} + 2^{i-1}\tau_1$, $2 \leq i \leq 8$. The subdomains are exponentially expanded as in Table 2. The reason for choosing the sub-domains this way is suggested by considering the typical patterns of lag pdf and jitter DLF; the lag pdf has a exponential decaying tail, which is true in our case, and the jitter DLF approaches a fixed limit.



(a) (b)

Figure 5: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *au_e* and its down sampled subtraces.



(a) (b)

Figure 6: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *us_e* and its down sampled subtraces.

In the experiments, we constructed the approximate DLF tables from the first-half-traces of trace *uk_p*, *ar_p*, *au_p* and *us_p*, and ran the DLF algorithm over the second-half-traces of the same traces and their down-sampled sub-traces, as in Scenario II. Jacobson's algorithm already ran over these traces in Scenario II. We only kept 8 rows in the DLF table. When a lag in question larger than τ_8, the table search of the DLF algorithm would simply return the $C_2(8)$.

Figure 11 - 12 shows the experimental results. In all cases the DLF algorithm in this scenario generates slightly larger boundary predictions than that in Scenario II, probably caused by the coarser lag resolution, but better correct prediction probabilities, roughly 1% gain on average over Scenario II. Based on the experiments it may be fair to draw the conclusion that the approximate DLF table does not degrade the performance of the DLF algorithm.

5 Implementation

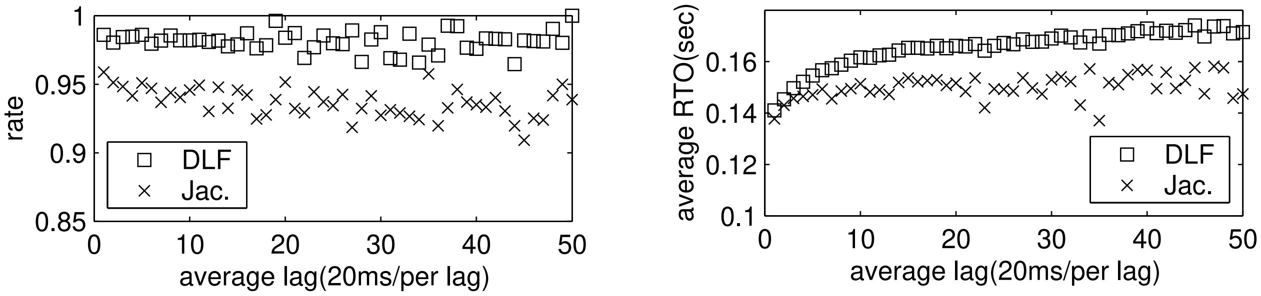
The advantage of Jacobson's algorithm is its low computing overhead and on-line implementation. The disadvantage of Jacobson's algorithm is its over-conservative initialization, which costs the algorithm at least 20 roundtrip iterations to converge to the true predictions, and its lack of robust statistics to be shared among connections across the same network path. Jacobson's algorithm does not work well with short-lived connections, in which the senders and receivers only exchange a small number of packets.

The advantage of the DLF algorithm is that the DLF table is robust, and can be shared by many connections across the same path. Also, each prediction always uses the most recent delay measurement as the prediction base, so it provides good adaptation to changing delay trends. The disadvantage of the DLF algorithm is the overhead of maintaining the table. The prediction itself is simply a table search, so the success or not of the DLF algorithm may depend on whether or not we can design a low overhead, heuristic algorithm to construct the table.

We present here a framework for implementing the DLF algorithm. The basic idea is that a host runs a process called Table Builder (TB) in the background (similar to the proposed Congestion Management process [2] for a host, or combine them together so they can share the common state variables); TB is responsible for constructing the DLF tables for each path. The transport protocols communicate with the TB to get the necessary table before they launch a communication session with a remote host. If the TB fails to provide a DLF table for a remote host, implying there has never been a communication between the two hosts, or they did not communicate with each other for a long time, the transport protocols can switch to a backup scheme, such as Jacobson's algorithm.

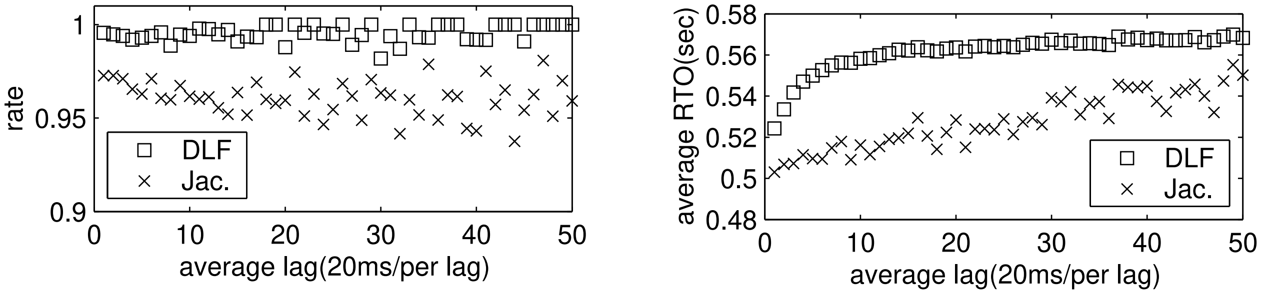
Issues on what data structure the TB process uses to store these tables or how to cache them in memory or store them in disk are beyond the discussion here. We concentrate on how a TB process can build a good DLF table for a remote network host.

There are two important issues related to building a good DLF table: the table size and the table updating scheme. Based on



(a) (b)

Figure 7: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *uk_p* and its down sampled subtraces.



(a) (b)

Figure 8: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *ar_p* and its down sampled subtraces.

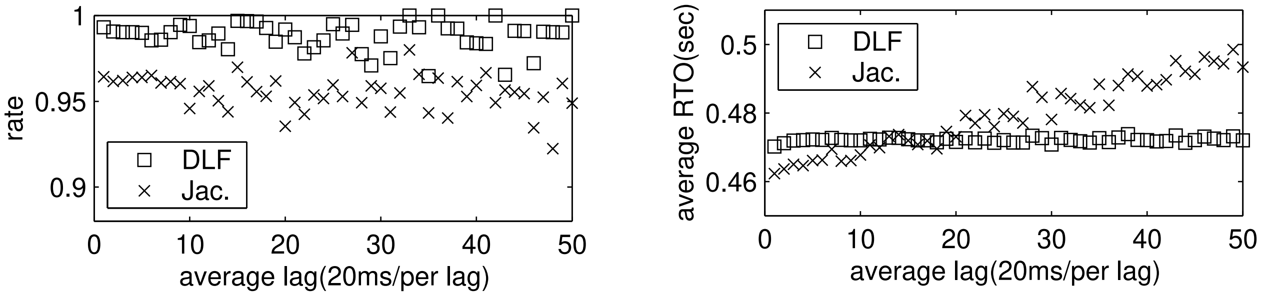
our experiments in Scenario III, we believe an approximate DLF table with $8 \times 2 + 1$ entries is good enough for the prediction. Here an extra entry beyond the last stores the mean delay of the path (not the SRTT) for initializing the algorithm. If the lag division scheme is fixed as in Scenario III, each table needs $8 + 1$ entries. If we need only a conservative coarse boundary prediction, $4 + 1$ entries may well serve the purpose, since in the tables built in Scenario III the last 4 entries were very close to each other. In the extreme we may need only $1 + 1$ entries for the table and it stores only the jitter deviation corresponding to the largest lag. Even with this simplification, the table can still give relatively better boundary predictions for short-lived connections than Jacobson's algorithm.

If the network traffic is stationary, the DLF table needs to be built only once forever. Unfortunately, network traffic in general is nonstationary (such as daily change trends); it may be considered as stationary only within certain period of time [17, 7]. To illustrate the nonstationary property of the network, Figure 13 shows how the DLF curves (in logarithmic scale) of the four paths studied above change with time. Each curve of the figure is calculated from a ping trace spanning over 200s, roughly with 10,000 samples (50 pings/sec). The four ping traces for each path were collected consecutively, but separated by 30 minutes. We think a trace with 10,000 samples is large enough to obtain a converged DLF curve. This figure shows that the DLF curves after 30 minutes do more or less shift away beyond the statistical error margins, clearly indicating a nonstationary process. Therefore, the TB needs to continuously update the DLF tables.

The updating scheme we propose has two parameters: the DLF table valid period (dlf_age), and the maximum number of jitter samples (max_num_jitter) needed for calculating each entry of the table. we suggest $dlf\_age = 15$ minutes, and $max\_num\_jitter = 200$. The TB process needs to maintain a queue of timestamped delay measurements for each path. Each new delay sample plus its timestamp will be inserted into the queue. After an inserting action all the delay samples with timestamps earlier than the newest timestamp by τ_8, the largest lag entry of the table, will be deleted from the queue (in this way the TB only maintains a relatively small window of delay measurements for each path); then the TB will start a new round of updating table entries. For any entry of the DLF tables if the TB process already counts enough jitter samples before the current valid period is over, it will stop updating this entry. When moving from one valid period to another, the TB process will not simply abandon all the entries. An entry can be aged according to the following formula:

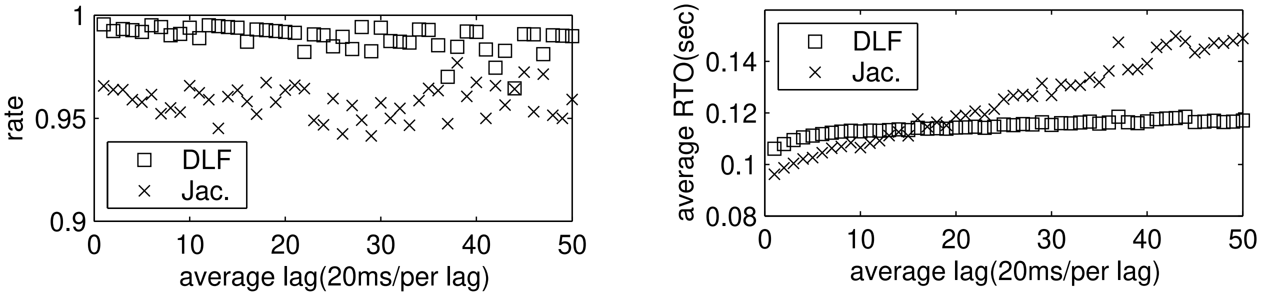
$$
\begin{aligned}
&new\_value = \\
&\qquad [\,(max\_num\_jitter - num\_sample) * old\_value \\
&\qquad + \sqrt{\sum_{n=1}^{num\_sample} jitter\_sample_n^2}\ ]/max\_num\_jitter
\end{aligned}
$$

where num_sample is the total number of jitter samples obtained by the TB process so far for a entry within the current valid period, old_value is the old entry value inherited from the previous valid period, new_value is the new entry value and $jitter\_sample_n$ is the nth jitter sample. This aging formula



(a) (b)

Figure 9: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *au_p* and its down sampled subtraces.



(a) (b)

Figure 10: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *us_p* and its down sampled subtraces.

will produce a smooth transition from one valid period to another and if the TB process can collect sufficient samples within a valid period, the entries will be completely updated. Obviously we can use a similar aging formula to update the mean delay entry.

The running time of the updating algorithm is

$$\mathbf{O}(num\_table\_entry * max\_num\_jitter / dlf\_age).$$

To deal with the scalability issue the TB process needs only cache the recently used paths, and save the remainder on disk, if justified. Considering that there can be a large number of TCP connections in a busy web server, we may need a garbage collection scheme to delete tables for very old non-active paths. Each table entry may also maintain a $record\_value$ field to store the largest entry values seen so far. When a path changes from non-active to active, the TB process can copy the $record\_value$ fields to the entries' old_value fields. This is a conservative way to initialize the table when a path goes from non-active to active.

The skeleton implementation presented here leaves many open issues for future studies, such as the initialization of the DLF table, definition of active and non-active path, passive (as we proposed here) versus active (a dedicated protocol based on ping) construction of the DLF table.

We strongly feel that a well tuned TB process is crucial to the success of the DLF algorithm in the future. More experiments show that, by choosing $k_d = 4$ for (18), the DLF algorithm can achieve a prediction error rate less that 0.5%. Even with $k_d = 4$, the DLF algorithm still gives a boundary prediction well below the initial value assigned by Jacobson's algorithm. For short-lived connections the DLF algorithm is definitely a winner. Since it has an architecture similar to the proposed congestion management (CM) algorithm, we may find ways to combine them together to further reduce the overhead of maintaining the DLF tables.

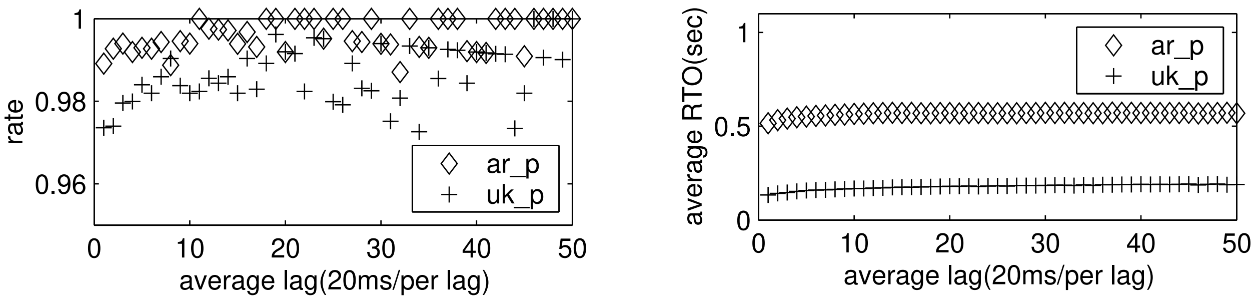
6 Conclusion

An improved understanding of the nature of wide-area network delays can help us to design better delay boundary prediction algorithms. By rethinking the model implicit in the conventional SRTT calculation, we learnt that SRTT is not a good mean estimator of RTT, but the best real-time estimate of RTT based on an ARMA model. The ARMA model suggests that for delay sequences showing a strong nonstationary properties, the SRTT calculation should choose a large parameter $g < 1$, but for white noise sequences, a small parameter $0 < g$ is required.

We developed a new model — a multi-structure model to explain the random properties of delay sequences, and a new algorithm based on the deviation-lag function (DLF) to predict delay boundaries. The algorithm applies the Chebychev inequality of probability theory. Experiments showed that the DLF algorithm outperformed Jacobson's algorithm roughly by a 60% improvement in average prediction error rate. It also adapted faster to delay changes than conventional algorithms derived from ARMA models. The DLF algorithm needs to build a DLF table in advance, introducing more computing overhead than that of Jacobson's algorithm. In order to show the possibility of doing a real

Table 2: The lag domain division scheme (in second)

$[0, \tau_1)$	$[\tau_1, \tau_2)$	$[\tau_2, \tau_3)$	$[\tau_3, \tau_4)$	$[\tau_4, \tau_5)$
$[0, 0.01)$	$[0.01, 0.03)$	$[0.03, 0.07)$	$[0.07, 0.15)$	$[0.15, 0.31)$
$[\tau_5, \tau_6)$	$[\tau_6, \tau_7)$	$[\tau_7, \tau_8)$	$[\tau_8, \infty)$	
$[0.31, 0.63)$	$[0.63, 1.27)$	$[1.27, 2.55)$	$[2.55, \infty)$	



(a) (b)

Figure 11: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *uk_p* and *ar_p* and their down sampled subtraces when using approximate DLF tables.

implementation, we presented a skeleton model for the DLF algorithm, which includes a Table Builder (TB) process running on the background. We designed a scalable and robust table entry updating algorithm for the TB. Since the architecture of the proposed skeleton is similar to that of the proposed Congestion Management algorithm, we suggested combining them together to reduce the overhead for building DLF tables.

7 Acknowledgments

The authors would like to thank the anonymous reviewers for their very helpful comments on an earlier version of this paper.

References

[1] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proceedings of ACM SIGCOMM'99*, pages 263–276, Cambridge, Massachusetts, USA, 1999.

[2] H. Balakrishnan, H. S. Rahul, and S. Seshan. An integrated congestion management architecture for internet hosts. In *Proceedings of ACM SIGCOMM'99*, pages 175–188, Cambridge, Massachusetts, USA, 1999.

[3] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, Englewood Cliffs, New Jersey 07632, 2nd edition, 1992. see page 211-213.

[4] M. S. Borella, S. Uludag, G. B. Brewster, and I. Sidhu. Self-similarity of internet packet delay. In *Proc. IEEE ICC'97*, August 1997.

[5] J. W. Cohen. *On regenerative processes in queueing theory*. Springer-Verlag, New York, 1976.

[6] J. Feder. *Fractals*. Plenum Press, New York, 1988.

[7] A. Feldmann, A. C. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of internet WAN traffic. In *Proceedings of ACM SIGCOMM'98*, 1998.

[8] C. A. Fulton and S. Li. Delay jitter first-order and second-order statistical functions of general traffic on high-speed multimedia networks. *IEEE/ACM Transactions on Networking*, 6(2):150–163, April 1998.

[9] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM'88*, pages 314–329, Stanford, CA, 1988.

[10] P. Karn and C. Partridge. Improving round-trip time estimations in reliable transport protocols. *ACM Trans. on computer Systems*, 9(4):364–373, November 1991.

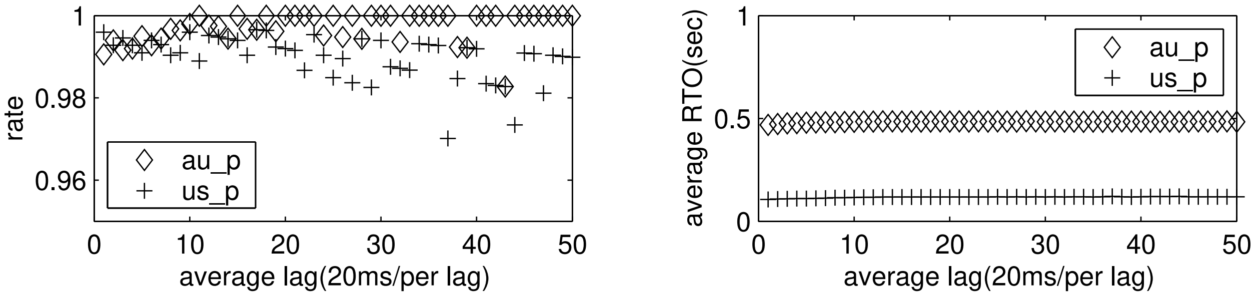
[11] Q. Li. *Delay Characterization and Performance Control of Wide-Area Networks*. Ph.D. dissertation, University of Delaware. Available at http://www.ece.udel.edu/˜qli, May 2000.

[12] Q. Li and D. L. Mills. Investigating the scaling behavior, crossover and anti-persistence of the Internet packet delay dynamics. In *Proceedings of IEEE Globecom'99*, Rio de Jonerio, Brazil, 1999.

[13] L. Ljung. *System identification: theory for the user*. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1987.

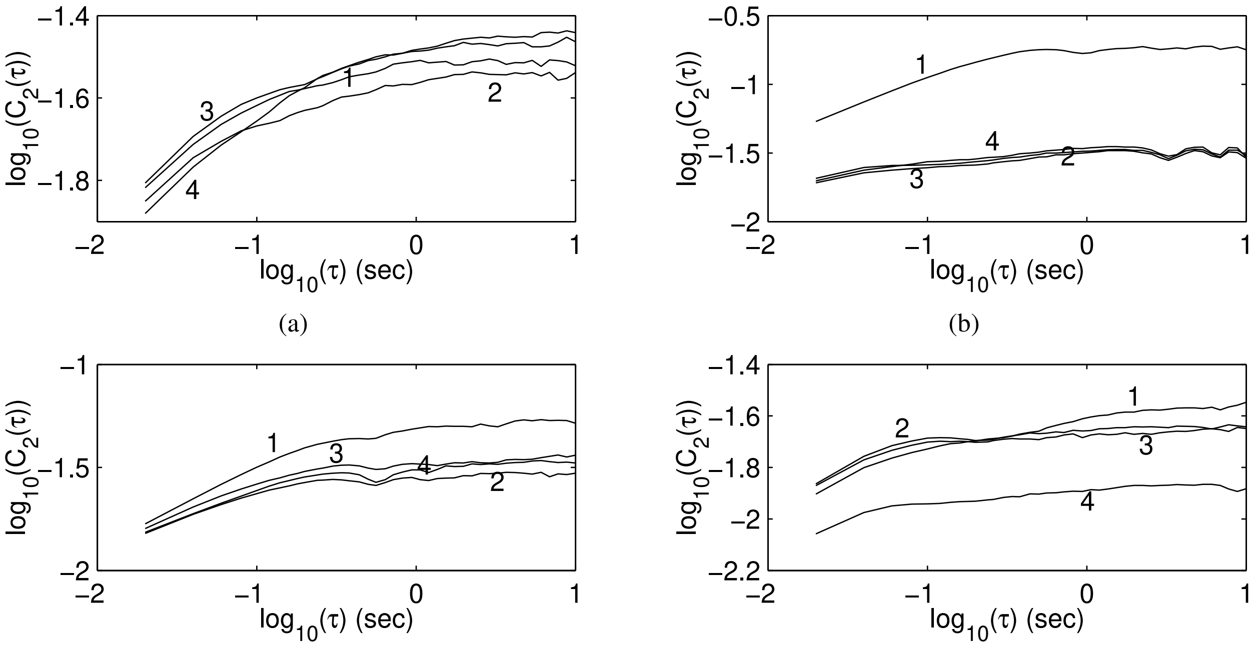
[14] P. Meakin. *Fractal, scaling and growth far from equilibrium*. Cambridge University Press, 1998.

[15] J. Postel. Transmission control protocol specification. RFC-793, SRI International, Menlo Park, CA, September 1981.



(a) (b)

Figure 12: The correct-prediction rates (a) and the average predicted delay boundaries (b) of trace *au_p* and *us_p* and their down sampled subtraces when using approximate DLF tables.



(a) (b) (c) (d)

Figure 13: The deviation-lag functions of paths (a)*uk*, (b)*ar*, (c)*au*, (d)*us*.

[16] A. Privalov and K. Sohraby. Per-stream jitter analysis in CBR ATM multiplexors. *IEEE/ACM Transactions on Networking*, 6(2):141–149, April 1998.

[17] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, January 1997.