# Bandwidth Modelling for Network-Aware Applications

J. Bolliger[1], Th. Gross[1,2] and U. Hengartner[1,2]

[1]Departement Informatik
ETH Zürich
CH 8092 Zürich

[2]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Abstract*— **Network-aware applications attempt to adjust their resource demands in response to changes in resource availability. E.g., if a server maintains a connection to a client, the server may want to adjust the amount of data sent to the client based on the effective bandwidth realized for the connection. Information about current and future network performance is therefore crucial for an adaptive application.**

**This paper discusses three aspects of the coupling of applications and networks: (1) a network-aware application needs timely information about the status of the network; (2) a simple bandwidth estimation technique performs reasonably well for TCP-Reno connections without timeouts; (3) enhancements proposed to TCP-Reno to reduce the number of timeouts (i.e., SACKs and its variants) increase the bandwidth but also improve the accuracy of bandwidth estimators developed by other researchers. The empirical observations reported in this paper are based on an in-vivo experiment in the Internet. Over a 6-month period, we logged the micro dynamics of random connections between a set of selected hosts. These results are encouraging for the developer of a network-aware application since they provide evidence that a simple widening of the interface between applications and network (protocol) may provide the information that allows an application to successfully adapt to changes in resource availability.**

## I. Introduction

Many of today's networks do not provide service guarantees. As a result, an application that maintains a connection between two hosts may experience varying performance during the lifetime of the connection. E.g., the bandwidth available to any one application depends on the demands of all other applications that happen to share a network component (link, router). *Network-aware* applications attempt to react to changes in network performance: such applications try to adjust their resource demands in response to actual resource availability. E.g., an application may transmit only a smaller version of an image if the network is congested.

Network-awareness provides a way to implement an end-to-end quality-of-service model for an application. That is, if the service model allows an application to trade off the quality of the transferred objects (e.g., size) for other parameters of the transfer (e.g., meeting a time limit), then a network-aware application can adapt to resource changes in the network. A network-aware application can, e.g., transfer a smaller (lower-quality) version of an object to meet the time limit.

The developer of a network-aware application must decide on a service model that fulfills the requirements of the problem domain. Only the application (developer) can decide what constitutes "high quality", and there are many application-specific quality models. In this paper, we concentrate on those that correlate quality with the size of the objects transferred, i.e., a larger object is assumed to have higher quality. (This model covers a large number of applications, e.g., access to image databases.) To make a selection between objects of different quality levels (sizes), the application requires information about shortage or availability of network resources. However, obtaining information about the current and future state of the network is far from easy. Therefore several researchers have developed network performance models that attempt to derive the actual (effective) bandwidth from various network and protocol parameters [13, 16]. If such models are realistic, then they are an attractive source of information for a network-aware application.

In this paper we report on the effectiveness of such network performance models. The paper starts with a brief review of network-aware applications (Section II) and a discussion of the impact of untimely information about the network status (Section III) and the necessity of accurate performance models. Section IV reviews the network performance models studied. Over a 6-month period, we performed an in-vivo experiment in the Internet to gather performance data for connections between a number of hosts in North America and Europe (Section V). We analyze the micro measurements of these connections to determine the accuracy of these models (Section VI) and the effectiveness of various enhancements to TCP and their influence on the accuracy of these models (Section VIII).

## II. Network aware applications

There exist a number of approaches to the design of network-aware applications. To focus the discussion of the requirements imposed by network-aware applications, we briefly sketch a framework for network-aware applications [3]. By focusing on a framework, we can address the concerns of a large class of adaptive applications, i.e. the class of network aware applications that are implemented by extending this framework.

This framework aims at the construction of interactive applications that can make a choice between transmitting higher or lower quality objects. The user sets a time limit to bound transfers; when network bandwidth deteriorates (e.g., when access-

Research sponsored in part by ETH Zuerich Polyprojekt 41-2641.5, and in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

ing a distant server), only low quality objects are transferred; when there is extra bandwidth, higher quality objects are delivered. To demonstrate the usefulness of the framework, it has been extended to implement Chariot, an image search and retrieval system [3].

The core of the framework is a *feedback loop* with 3 phases: the *monitor and react phase*, which takes information from lower layers to decide if the current bandwidth is sufficient to complete the current transfer in the alloted time frame; the *prepare phase*, which takes objects that require an adjustment of their quality state and produces either a higher-quality or a lower-quality object, and the *transmit phase*. The monitor phase wants to know the amount of data $d$ that can be transmitted in the interval $[now, time\_limit]$ (i.e., the integral of the available bandwidth over this interval); if $d$ exceeds the amount of data left to transfer, then higher quality versions can be admitted. Otherwise lower quality versions must be selected or generated on-the-fly (e.g., by compression).

The key observation is that the monitor phase is interested in the bandwidth available to the application. Network-centric metrics like the number of competing connections at the bottleneck router are of limited assistance. This specific framework presents one path to the development of network-aware applications; there exist applications developed without a framework as well as other frameworks [21]. However, all adaptive applications must make decisions based on the (actual, reported, or estimated) current and future network performance. So we concentrate in this paper on how to obtain (without delay and at low cost) a value for the network performance that can drive application adaptivity.

### A. Related work

The focus of this paper is to investigate TCP bandwidth models for use in network-aware applications as well as the influence of various enhancements to TCP on the accuracy of these models.

A review and assessment of network-aware applications is beyond the scope of this paper, we refer to [3].

We postpone a discussion of the various models for throughput until Section IV-B.

We assume that the reader is familiar with recent enhancements to TCP: Hoe's proposal [6], as well as selective acknowledgements (SACKs) [12], and an enhancement to TCP's congestion recovery algorithm termed FACK (forward acknowledgement) [10, 11] aim at solving the problems caused by multiple packet loss in a single congestion window. The SACK enhancements also allow the detection (and repair) of lost fast retransmissions [11]. Lin et al. [9] and Balakrishnan et al. [1] independently proposed a window optimization to further reduce the number of timeouts that complements the SACK-based TCP enhancements.

## III. Timely feedback

Timeliness and accuracy of the information about the available bandwidth are important prerequisites to a useful prediction of future bandwidth. To understand the possible impact of untimely and inaccurate predictions of the bandwidth available, we conducted the following experiment:

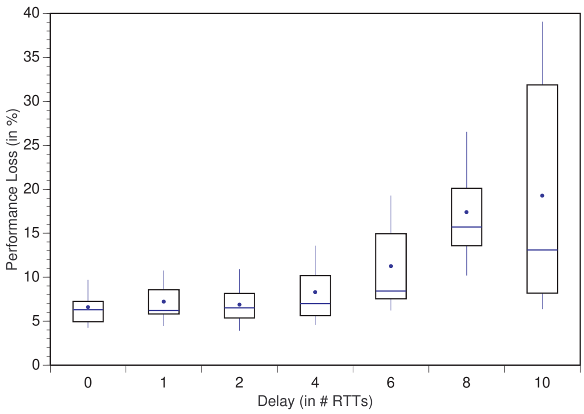


Fig. 1. Box-plot of performance loss as function of delay in obtaining bandwidth information.

We trace the transfer of a set of images that are returned by a Chariot server [3] in response to a query. The source of the transfer is located at ETH Zürich (Switzerland), the sink at the University of Linz (Austria). The bandwidth trace, which exhibits considerable variation, as well as the evaluation of the framework's ability to dynamically adapt to the changes in available bandwidth have been reported in [3] and are omitted here due to space limitations. To ensure reproducibility of the experiments, we use trace modulation [15] to perform an application-transparent emulation of the Internet transfer on our LAN. The application traffic is delayed according to delay and bandwidth parameters read from the replay trace collected beforehand.

To study the effect of untimely (hence potentially inaccurate) bandwidth information, we rerun the same experiment varying the amount of delay introduced in the bandwidth estimates. A delay of $n$ round-trip times (RTT) means that the application learns about the currently available bandwidth only after $n$ RTTs[1]. We investigate different levels of delay: 0 (which is the reference behavior as reported in [3]), 1, 2, 4, 6, 8, and 10 round-trip times; the average RTT for the transfer was approx. 100 ms. For each level, we conduct 40 experiments under the same conditions as the example above and summarize the performance penalty introduced by the delayed bandwidth information in the box plot in Figure 1. The $x$-axis shows the level of delay, the $y$-axis represents the relative performance loss witnessed by the adaptation process: $100 \cdot (possible - actual)/possible$, where *possible* denotes the amount of data that could have been transferred within the time limit set by the user, and *actual* reflects the amount of useful data that was actually delivered to the user. The top and bottom line, and the line through the middle of the box correspond to the 3rd quartile, 1st quartile, and median respectively; the whiskers on the bottom and top extend from the 10th percentile to the 90th percentile; the dot indicates the mean.

The figure shows that out-dated information on the bandwidth available does *not necessarily* result in lower goodput; the 10th percentiles of the experiments with 0–6 and 10 RTTs delay are all in the range of 6–9 %. However, untimely and hence potentially wrong estimates on the bandwidth available *can* increase

[1] Delayed bandwidth information *may* be inaccurate (e.g., if bandwidth changed within the $n$ RTTs), but *need not* be inaccurate (e.g., if bandwidth remained unchanged).

the number of situations that require adaptation, and therefore can increase the overhead of the decision making. Furthermore, the more often adaptation is required, the higher the risk that we encounter limited adaptation potential or problems with the scheduling of the *prepare* and *transmit* phases [3]. A high number of situations that call for adaptation may eventually result in a higher penalty in terms of lost opportunities to transmit. Untimely information therefore *can* increase the risk of performance loss, as is illustrated by the data for both the 3rd quartile and the 90th percentile. These values increase considerably with delayed bandwidth information.

A detailed discussion of all aspects of the interaction between network protocols and adaptivity of an application is beyond the scope of this paper. However, even this simple experiment indicates that timely and accurate information about application-relevant metrics is important for network-aware applications. We therefore concentrate in this paper on the modelling and prediction of bandwidth.

## IV. Approaches to bandwidth modelling

There are basically two aspects of bandwidth prediction as needed by network-aware applications. First, since decisions on adaptation must be carried out dynamically, i.e. at run-time, we need a way to dynamically assess the bandwidth currently available to the application. Second, we are interested in an estimate of the bandwidth available in the (near) future, given a (short) history of bandwidth samples.

In this paper we concentrate on the first aspect and compare models that help to characterize the bandwidth available. We touch only briefly on problems related to the estimation of future bandwidth.

### *A. Classification*

There exist three basic approaches to collect information about the bandwidth that is currently available to an application. These approaches can be distinguished according to the layer in the protocol stack that they use to acquire information:

- Application-level bandwidth monitoring may be a straight-forward approach, but since layering prevents getting additional timely information about lower layers, this approach cannot distinguish between transient behavior (timeouts, slow-start, ...) and low bandwidths for instance due to congestion.
- Transport-level information is used by congestion-aware (and therefore adaptive) transport protocols, which have most of the information needed readily available, e.g., in the case of TCP, the congestion window (*cwnd*), round-trip time (*rtt*), number and duration of timeouts, etc. A recently developed model [16] uses this information to characterize TCP throughput.
- Network-level information (such as loss-rates, packet-timings, etc.) can also be used to derive application-relevant bandwidth metrics, e.g., the bottleneck bandwidth [17], as well as transport level throughput [13].

Measurements at the application layer are hardly a timely and reliable indicator of available bandwidth, due to the high overhead associated with gathering the data, and are not discussed further. We now try to address the question if the information maintained by the protocol layers is of any use or help in modelling and predicting bandwidth.

### *B. Throughput models*

Throughput models try to characterize the TCP throughput as a function of a given set of (stable, non-changing) parameters. There are several properties of a model that are of importance in the context of network-aware applications. First, the model should be accurate. However it is also important that the model can be computed efficiently. The fewer parameters are needed, the better: a smaller number of parameters implies that fewer parts of the protocols are affected.

We consider three simple models: one ($P_1$) is directly derived from the TCP specification, the other two ($P_2$, $P_3$) have recently been described in the literature [13, 16].

$P_1$ TCP's measure of the available bandwidth is given by the size of the congestion window and the measured RTT [7, 18]. Thus a straight-forward approach is to simply use the averages of *cwnd* and *rtt* to characterize the average application-level throughput.
Pros: Simple to compute, exposing the parameters requires only small interface changes.
Cons: Applicable only to TCP, may be too simple to be accurate.

$P_2$ [13] models TCP congestion avoidance behavior (i.e. TCP throughput) based on loss-rate $p$, the round-trip time $rtt$, and the packet size *mss*.
Pros: (i) Mainly dependent on network-layer metrics (hence applicable for purposes other than modelling TCP throughput, e.g., conformance testing for other congestion-aware protocols (e.g., reliable multicast [20]) or bandwidth allocation in the Internet); (ii) model is based on a minimum of information, it is simple to compute.
Cons: (i) The model attempts to capture asymptotic behavior of the TCP congestion avoidance algorithm (and thus assumes that packet loss must occur) and may therefore not be suited for brief connections or connections without loss; (ii) accuracy may be low, the model does not consider timeouts.

$P_3$ [16] extends $P_2$ by modelling timeouts and connections whose throughput is limited by the receiver window. In addition to the loss-rate, the round-trip time, and the packet size, this model includes the number of timeouts, the average duration of a retransmission timeout, and the size of the receiver window.
Pros: Since the model is based on a more comprehensive set of parameters, the accuracy should be high.
Cons: (i) The model is based also on transport level metrics, such as the number of timeouts and the average duration of a timeout, instead of using only network-layer metrics (such as loss rates); (ii) although the model is still fairly simple, it needs more information (which is harder to get) than $P_2$; (iii) it models asymptotic behavior of congestion avoidance and may not be suited for short-duration connections (same argument as for $P_2$).

## V. AN INTERNET EXPERIMENT

To generate a collection of traces that we can use to validate (or refute) our conjectures about the network requirements of adaptive applications, we performed an in-vivo experiment in the Internet. Here we briefly summarize the setup of the experiment, see [4] for details.

We produced a family of user-level protocol implementations to study the effectiveness of (standard) TCP (Reno) congestion control as well as to evaluate various enhancements to recover from or avoid congestion [4]: Vegas, SACK/FACK, Rate-Halving, and Rate-Halving with Lost Retransmission Detection (LRD). We then installed a daemon [17] at a number of Internet hosts in North America (CMU, MIT, UC Irvine, DEC SRC Palo Alto, U Waterloo (Canada)) and Europe (ETH Zürich, Berne (Switzerland), Linz (Austria), Munich (Germany), Madrid (Spain), Turku (Finland)). From time to time, the daemons at randomly selected source and sink hosts were contacted and instructed to carry out transfers of 1 MB of data using a randomly chosen protocol variant. (To ensure that the receiver window does not limit the bandwidth available to a connection, we chose large windows.) The intervals between successive transfers between a given pair of hosts are taken from an exponential distribution with a mean of 10 hours. We recorded detailed measurements of the micro-dynamics of each connection. These daemons were active over a 6-month period (from November 1997 till May 1998). The experiment includes the records of 24943 connections: 4148 Reno, 4100 Vegas, 5415 FACK, 5569 Rate-Halving, and 5711 LRD connections.

## VI. EVALUATION OF MODELS

The usefulness of prediction models for the needs of network-aware applications rests on the accuracy of the information: How well does a model capture the bandwidth currently available? If the model passes the accuracy requirements, then we should also consider the stability, i.e. for how long are (good) predictions likely to be valid?

We address both concerns in turn. Both aspects have been individually studied by previous research; [13, 16] studied the accuracy of their models, while [2, 17] analyzed the temporal stability of metrics such as available bandwidth, loss rates, etc. We provide new insights based on a new set of measurements and discuss to what extent these models can be used in the context of network-aware applications (which may have to adapt after shorter periods than those assumed in earlier studies, which concentrated on computing the asymptotic bandwidth).

### A. Accuracy of models

For each connection of our experiment and each of the models $P_i$, we compute the throughput $\beta^{(i)}_{predicted}$ as predicted by the model.

For $P_1$ we compute the weighted average of the number of packets the connection kept in flight, i.e. $\overline{cwnd}$ as the weighted average of the *cwnd* size over the duration of the whole connection, where the weights are given by the time *cwnd* stayed constant. $\beta^{(1)}_{predicted}$ is computed as the ratio of $\overline{cwnd}$ and the mean of the round-trip time samples produced by the protocol $(\overline{rtt})^2$.

We compute $\beta^{(2)}_{predicted}$ according to the formula developed in [13] as a function of $p$ and $\overline{rtt}$. $p$, the loss indication probability is defined as $\frac{TO+TD}{dataSent}$; $TO$ is the number of timeouts, and $TD$ is the number of triple duplicate acknowledgement events experienced that lead to fast retransmit/fast recovery. *dataSent* is the total number of packets sent (i.e. including retransmissions).

$\beta^{(3)}_{predicted}$ is computed using the "approximative" model developed in [16]. Since our connections were not receiver-window limited, we compute $\beta^{(3)}_{predicted}$ as a function of only $\overline{rtt}$, $p$ and $T_0$. $T_0$ represents the average duration of a (non-backoffed) retransmission timeout.

Let $\beta_{measured}$ be the throughput as experienced by the application. We can use a scatter plot of the pairs $< \beta_{predicted}, \beta_{measured} >$ to verify the accuracy of a model by checking whether a clear linear relation is discernible or not. Furthermore we attempt to statistically characterize the "goodness" of a model by computing the coefficient of determination $R^2$ for a simple least-squares fit of the data. ($R^2$ is the square of the sample correlation and denotes the fraction of the variation that is explained by the regression [8]).

Figure 2 depicts these pairs $< \beta^{(1)}_{predicted}, \beta_{measured} >$ for $P_1$. Clearly, $P_1$ is far from being close to an accurate model. In many cases, it grossly overestimates the bandwidth achieved. This deviation is not surprising since the $\overline{cwnd}$ also includes the idle periods before retransmission timeouts. We clustered the samples according to the number of timeouts experienced by a connection ($TO = 0, 1, 2$, and $TO \geq 3$). We tried to fit the clusters with quadratic functions simply to visualize the tendency of the overestimation to increase with $TO$. Space limitations do not permit to show the scatter plots for $P_2$ and $P_3$. However, a similar tendency can be observed when looking at the data for $P_2$; again this result is not surprising either since the model does not take retransmission timeouts into account [13]. In contrast to $P_1$ and $P_2$, which both overestimate the measured throughput considerably (in the presence of timeouts), $P_3$ seems to model the measured data well, independent of the number of timeouts experienced. This finding is consistent with the results reported in [16].

The observations reported are also supported by the simple statistical analysis that is summarized in Table I. This table shows a statistical comparison of the models for Reno connections based on different clustering criteria of the data. The table confirms that timeouts are problematic from a modelling and hence also from an application perspective, since $R^2$ reaches an high level for the simpler models merely for connections that do not suffer from timeouts. Table I also illustrates that we should exercise caution when summarizing the "goodness" of a model by a single number: note that according to the $R^2$ values, $P_3$ seems to predict the "lossy, timeouts" cases better than "lossy, no timeouts" cases, which is somewhat surprising and will be explained shortly. Note that $P_2$ and $P_3$ are not applicable to loss-free connections.

[2] We account for the artificial inflation of *cwnd* in fast recovery [18, 19] by only using the constant number of packets in flight during recovery instead of the actual value of *cwnd*.

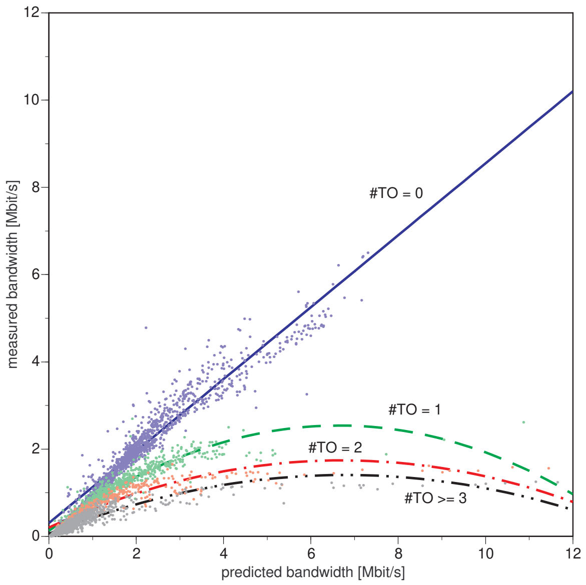


Fig. 2. $P_1$: all data points for Reno connections; clustered according to the number of timeouts per connection.

| $R^2$ | | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|---|
| all connections | | 0.74 | – | – |
| no loss | $(p = 0)$ | 0.99 | – | – |
| lossy | $(p > 0)$ | 0.62 | 0.83 | 0.92 |
| lossy, no timeouts | $(p > 0 \wedge TO = 0)$ | 0.99 | 0.85 | 0.85 |
| lossy, timeouts | $(TO > 0)$ | 0.63 | 0.86 | 0.93 |

TABLE I
SUMMARY OF MODELS (RENO CONNECTIONS).

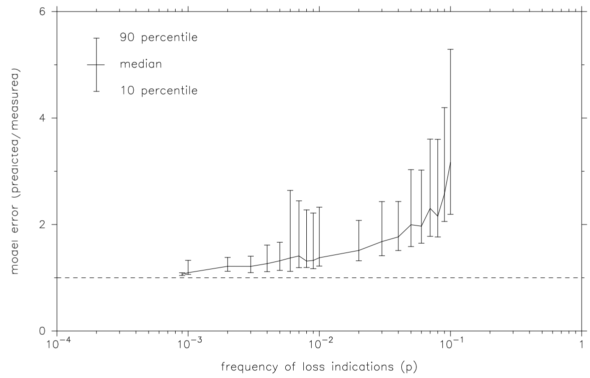


Fig. 3. $P_1$: model error as a function of loss indication frequency ($p$) for lossy Reno connections.

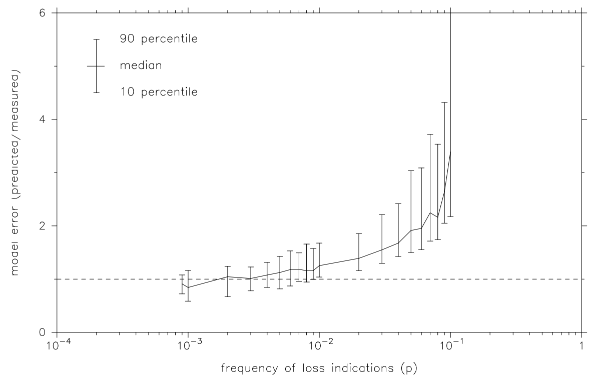


Fig. 4. $P_2$: model error as a function of loss indication frequency ($p$) for lossy Reno connections.

There is a simple way to improve $P_1$: we can adjust the computation to take the number of timeouts and the average duration of a timeout ($T_0$) into account. I.e., the average *cwnd* size is reduced by $(\overline{cwnd} \times TO \times (T_0 - \overline{rtt})/T_0)$. For this variation we obtain $R^2 = 0.94$ for all connections. ($R^2$ then is 0.86 for all lossy connections, 0.77 for lossy connections with timeouts; the other values remain unchanged.) However, even with this adjustment, $P_3$ is clearly more attractive than the other models.

To gain further insight about the behavior of the models for different loss rates, we plot the relative model error, i.e. the ratio of predicted and measured bandwidth, for the range of loss indication probabilities $p$ encountered. Figures 3, 4, and 5 show the graphs for $P_1$, $P_2$, and $P_3$ respectively. The $x$-axis represents the frequency of loss indications ($p$) on a log-scale. The $y$-axis shows median and the 10 and 90 percentiles of the relative model error. Note that $y$ values of 2 and 0.5 represent both an error of a factor 2. (The figures may therefore not be immediately intuitive, but we want to depict that these models rather overestimate than underestimate the true bandwidth.)

As can be clearly seen when comparing these graphs, $P_3$ outperforms the other models for connections with a high loss frequency. However, we still observe a non-negligible spread for $P_3$, which is slightly increasing towards small values of $p$. We note that connections with small $p$ are generally connections with no or very few timeout events. This observation allows us to resume the discussion of the statistical analysis, where we noted that connections suffering timeouts were better matched by $P_3$ than connections without timeouts.

By clustering the $P_3$ data points (connections) according to continents (see Table II) and identifying the distributions of the measured throughputs for Reno connections for these clusters (Figure 6), we see that for the low-bandwidth inter-continental connections (AE and EA), the prediction is better than for the higher-bandwidth intra-continental connections (AA and EE) (see Table II).

This result indicates that the quality of the model is closely related to the duration of the connection: the models $P_2$ and $P_3$ were developed to characterize asymptotic TCP throughput, based on the assumption that the time a connection spends in

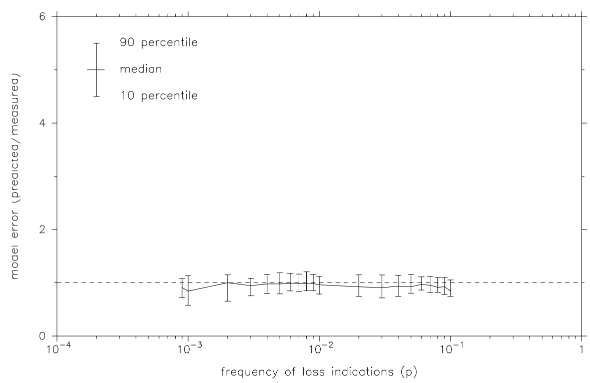


Fig. 5. $P_3$: model error as a function of loss indication frequency ($p$) for lossy Reno connections.

|  | median $\beta_{measured}$ [Mbit/s] | $R^2$ |
|---|---|---|
| $A \to A$ | 0.88 | 0.89 |
| $A \to E$ | 0.21 | 0.95 |
| $E \to A$ | 0.35 | 0.91 |
| $E \to E$ | 1.08 | 0.87 |
| all connections | 0.56 | 0.92 |

TABLE II

REGIONAL VARIATION FOR $P_3$ AND LOSSY RENO CONNECTIONS (A: NORTH AMERICA, E: EUROPE).

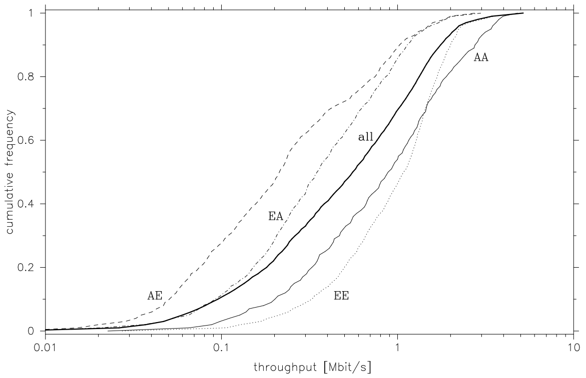


Fig. 6. Bandwidth distribution ($\beta_{measured}$) for lossy Reno connections; clustered according to continents.

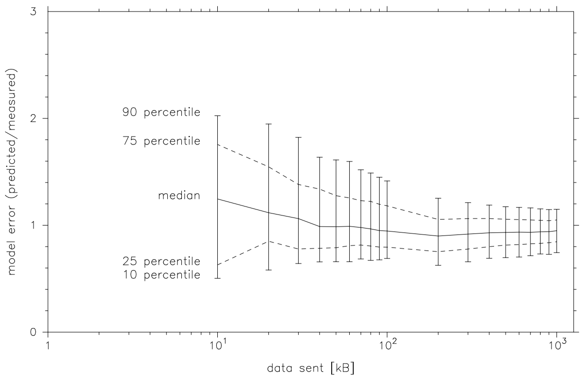


Fig. 7. $P_3$ model error as function of data transferred.

initial slow-start is negligible compared to the duration of the connection. Hence, it is not surprising to see that the accuracy of the model decreases as the connections get shorter (i.e. as the bandwidth increases for our fixed-size transfers). Despite this limitation, it is surprising to see how well $P_3$ still predicts the bandwidth; after all, 50 % of the connections we traced lasted less than 10 seconds.

Knowing about modelling errors at the end of a connection is only of limited usefulness to network-aware applications. Such applications need accurate and timely information as early as possible. The "box-plot" in Figure 7 shows how the model error of $P_3$ decreases as a transfer progresses. For each data point $x$ on the (logarithmically scaled $x$-axis), we applied $P_3$ to model the throughput experienced by the connection during the first $x$ kB of data sent. By virtue of $P_3$, only those connections were considered which experienced loss in the first $x$ kB of data sent. The figure indicates that the prediction provided by $P_3$ is fairly accurate even early on in "lossy" connections — which are the most difficult to model — as for 80 % of the connections the model differs by less than a factor of 2 (i.e. lies between 0.5 and 2).

*B. Stability of bandwidth*

Since at least one of the models ($P_3$) yields reasonable results, it is interesting to turn to an investigation of the stability of these data. For this aspect we can rely on earlier studies [17, 2]. The first study investigated the evolution of loss rates. This aspect is important for models $P_2$ and $P_3$, and the study discovered that the probability of seeing a zero-loss connection after having witnessed a zero-loss connection is approx. 0.75; this probability is fairly stable over time-scales of $10^2 - 10^4$ seconds. The same applies for non-zero-loss connections: such connections are again a good predictor for seeing a successive non-zero-loss connection. Furthermore, the available bandwidth is also a good predictor for the same time scales.

The more recent study [2] identifies two kinds of temporal evolution paths of the bandwidth for a particular sender-receiver pair: stable and unstable. For the class of host-pairs with stable bandwidth, the authors note that bandwidth is within a factor of 2, even for time intervals of up to tens of minutes ($10^3 - 10^4$ seconds).

The time-scales investigated by these studies may be too large for some network-aware applications. Even though available bandwidth may fluctuate (more) rapidly on smaller time-scales, the findings are encouraging for caching bandwidth information which may be appropriate, e.g., to deal with server selection or to help alleviate start-up problems of network-aware applications [3].

A preliminary analysis of our experimental data seems to support the findings of previous work (for smaller time-scales: $10^0 - 10^2$ seconds): We first look at how well the model for the first 50 % of the data transferred predicts the performance observed for the second half of the connection. Figure 8 shows the cumulative distribution function of the factor of bandwidth change observed (which in our simple prediction scenario is equivalent to the relative error of the prediction). We find that 84 % of the Reno connections experienced a change of less than a factor of 2 (to the better or the worse). Note that the figure does not show the complete distribution, as approx. 1 % of the connections experienced a throughput change of more than a factor of 4 (up to a factor of 100 !).

Network-aware applications need predictions about the bandwidth available in the future (see Section II). Such predictions must be based on a varying amount of past observations, and may well be required to predict the performance for varying time-scales into the future. Thus we are interested in assessing how the accuracy of a prediction depends on the amount of "history" data observed and the "future" to be predicted. Figure 9 shows (on the $y$-axis) the ratio of bandwidth change (or relative prediction error) as a function of the "history" observed and (on the $x$-axis) the "future" predicted. We show the median, 80 and 90 percentiles of the prediction error for each combination of the "history" and "future" values chosen[3]. The figure reflects

[3]For each point in the trace and each combination of the parameters "history" and "future" applicable, we determine the relative prediction error and aggregate

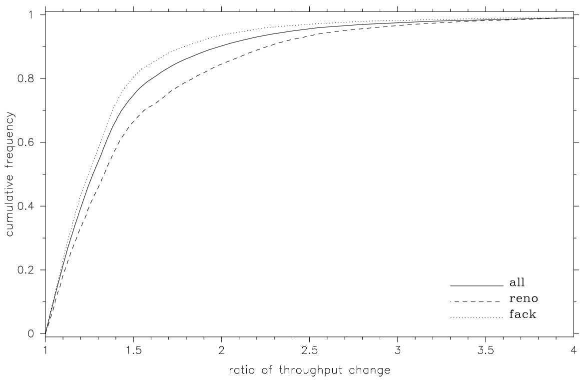


Fig. 8. Throughput change between first half to second half of connection.

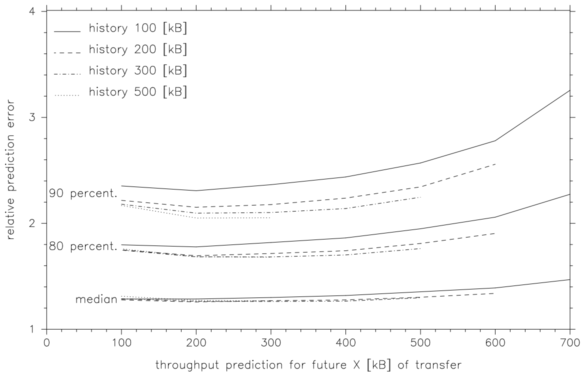


Fig. 9. Throughput change as function of history and future.

two obvious facts: (i) the more history data is used for the prediction the smaller the prediction error, and (ii) the further into the future we want to predict the less accurate the prediction is. What is more surprising to see, however, is that for most connections (i.e. 80 %) the dependence on the "history" and "future" parameters seems to be fairly small. This result indicates that even with medium level of past information, reasonable predictions about future performance seem to be possible (at least for the connections we observed).

Furthermore, it is interesting to note, that the prediction errors (stability of the throughput) observed seem to depend on the position within a connection. Figure 10 plots the same data for the prediction error for a fixed parameter "history" (100 kB) and different levels of "future" as a function of the position within a connection. We find that the prediction error decreases as the connection progresses (towards steady-state). This indicates that more work is required to understand the dynamics and performance implications of slow-start with respect to network-aware applications.

### C. Discussion

The results reported for $P_3$ in Section VI-A encourage the use of network and transport-layer information in modelling network performance. They provide a big incentive to pass this information to the application (or a framework upon which the

all the values for a "history"/"future" pair. Note that we exclude the slow-start from the "history" data. Slow-start's transient behavior dilutes findings about TCP steady-state behavior significantly and requires separate treatment.

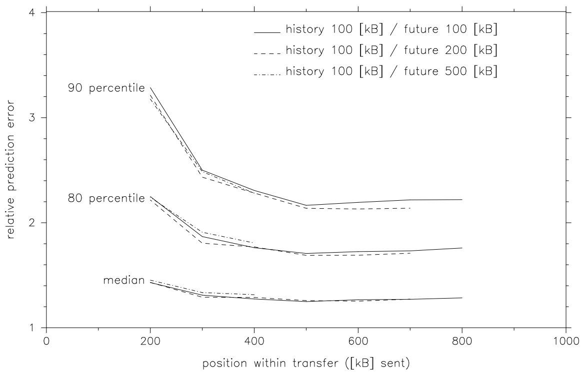


Fig. 10. Throughput change as function of history, future, and position within connection.

application is built). Other researchers have argued that the standard protocols should be *translucent* to expose QoS parameters. We take no position on the mechanisms that should be used to expose this information; our evaluation provides just another data point in favor of providing information from the lower levels of the protocol stack to the network performance modeller and predictor.

## VII. Timeouts

It is widely agreed that retransmission timeouts (RTOs) are problematic for various reasons. Besides causing modelling headaches and therefore unpredictable protocol behavior for network-aware applications [13, 16], they may waste network resources due to unnecessary retransmissions during slow-start after the timeout. Unfair sharing of the bottleneck and burstiness [14, 9], as well as performance losses experienced by the application [1], are other sources of concern.

Thus, it would be highly desirable if timeouts could be avoided as often as possible. To do so, we must understand the circumstances in which timeouts in TCP Reno happen. Following an earlier classification [9], we distinguish 3 scenarios:

*Multiple losses:* Multiple packets in a single congestion window are lost, and multiple fast retransmission/fast recovery cycles occur until the flow of duplicate acknowledgements ebbs of, leading to a RTO.

*Non-trigger:* The sender retransmits a packet without previous attempts because the fast retransmission algorithm has not been triggered.

*Lost retransmission:* The sender retransmits a packet that has already been sent in a preceding RTO or a fast retransmission.

Over the last few years, numerous TCP enhancements have been proposed that try to avoid timeouts and address the problems mentioned above. Selective acknowledgments (SACKs) [12] and FACK (forward acknowledgement) [10, 11] aim at solving the problems caused by multiple packet loss in a single congestion window. The SACK enhancements also allow the repair of lost fast retransmissions [11]. Studies of packet traces [9] have led to the conclusion that multiple packet losses are the source of only a very small fraction of the retransmission timeouts encountered by TCP connections and that over 85 % of RTOs are due to non-trigger of fast retransmission.

The data presented in the studies cited above were collected in 1995 and 1996. Although our data is not new in the sense that it points out new aspects of TCP behavior, the rapid evolution of the Internet (towards higher bandwidths for many links) introduces enough changes to warrant the experiment. Using a 3-month period of traces, we inspected the retransmission timeouts and their causes for all the Reno connections. We find that the 2486 Reno connections (from this subset) suffered 23279 timeouts, of which 39.5 % are due to non-trigger, 40.1 % are due to multiple packet loss, and 20.4 % are due to lost retransmissions (10.6 % lost fast retransmissions, 9.8 % lost timeout retransmissions). To get a handle on the benefits of SACKs, we installed receivers generating SACKs for Reno senders (which do not handle SACKs) and then traced the SACKs at the sender. This setup allows us to assess how many of the non-trigger timeouts could have been avoided by exploiting the SACK information (see extended recovery trigger condition [10]). It turns out that in our experiment, 32.9 % of the non-trigger timeouts would have been avoidable with SACKs. For the other 67.1 % of the non-trigger timeouts, optimizations such as NetReno [9] might have been effective.

To summarize, out of all the Reno timeouts observed, 63.7 % might have been avoided with SACK-enhanced protocols[4]. Note that this is an upper bound on the number what *might* be curable with SACKs. In Section VIII we report on how effective the SACK enhancements *are* in avoiding retransmission timeouts. In the other 36.3 % of the cases SACKs would not have been helpful. For the remaining 26.5 % non-trigger timeouts (not avoidable with SACKs), approaches such as NetReno would be applicable[5]. The numbers captured by our experiment largely differ from numbers reported in previous studies: 85 % of the timeouts are reported non-trigger situations in [9], and only 4 % of the timeouts have been classified as being avoidable by SACKs [1]. These differences can be attributed to the fact that the connections we traced observed bandwidths that are (at least) an order of magnitude higher as those reported in [2] and [17] (which are based on the same data as the studies cited above). This change may reflect the general trend towards higher bandwidths but may also be due to the fact that the sites participating in our study are generally well connected to the Internet.

## VIII. Evaluation of protocols

After assessing the potential benefits of various TCP enhancements with regard to timeout avoidance, we now compare the implementations of these protocol enhancements based on the actual measurements. We first report on the effectiveness of SACK-enhanced protocols in avoiding timeouts and then resume our analysis of predictors to see the effect of using different protocol implementations[6].

[4] 63.7 % = 40.1 (multiple loss) + 10.6 (lost fast retransmissions) + 0.329 * 39.5 (avoidable non-trigger) %.

[5] 26.5 % = 0.671 · 39.5 (unavoidable non-trigger) %

[6] Due to space limitations we mainly concentrate on the comparison of Reno and FACK connections. The data for Rate-Halving (with optional Lost Retransmission Detection) are similar to those for FACK. We have not yet included Vegas connections in our analysis. See [4] for more details.

### A. Effectiveness in avoiding timeouts

Overall, FACK is fairly effective in reducing the number of timeouts experienced. 67.9 % of the Reno connections suffer at least one timeout. The mean number of timeouts for Reno connections is 6.81. Only 34.1 % of the FACK connections are timeout-afflicted. The mean number of timeouts for FACK connections is 3.52, which is a 48.3 % improvement over Reno. Corresponding to the three classes of timeouts, there are three mechanisms present in SACK-enhanced protocols to avoid timeouts. We look at their effectiveness in turn (see [4] for details):

*Multiple loss:* When looking at timeouts due to multiple packet loss, we find that compared to 65.4 % Reno connections only 18.3 % FACK connections suffer from such timeouts, and the mean number of such timeouts per connection drops from 2.46 (Reno) to 0.42 (FACK). We conclude that FACK can drastically cut down the number of timeouts due to multiple loss. By and large, this reduction is achieved by the use of SACKs; the contribution of the other optimizations in the FACK algorithm is only minor in this respect.

*Non-trigger:* The FACK algorithm extends Reno's recovery trigger condition beyond the dupAck-threshhold logic to better cope with multiple packet and/or dupAck loss, which is particularily problematic for connections with small *cwnd* [10]. We find that 38.3 % of the Reno connections suffer from at least one non-trigger timeout. The mean number of non-trigger timeouts for Reno connections is 2.78. Only 19.2 % of the FACK connections suffer from at least one non-trigger timeout. The mean number of non-trigger timeouts for FACK connections is 1.75. The difference in means quite closely matches the reported fraction (approximately one third) of non-trigger timeouts that were termed "avoidable by SACKs" in Section VII (the difference is due to the fact that only a subset of the traces is used to assess the potential benefits of SACKs).

*Lost retransmissions:* Both Reno and FACK can recover from a lost retransmission only by a timeout. Lost retransmission detection [11] extends the FACK algorithm to avoid such timeouts; it achieves a 40 % improvement over FACK.

*Effect on throughput.* Figure 11 provides insight into how FACK's ability to avoid timeouts relates to performance seen at the application level. FACK performs consistently better than Reno as can be seen from the clear "right shift" of the bandwidth distribution curve.

### B. Effect on throughput models

Following the methodology used in Section VI, we try to assess the quality of the 3 models $P_1$, $P_2$, and $P_3$ for FACK connections by fitting the data with a least-squares fit and by computing the coefficient of determination $R^2$. Table III repeats the results for Reno from Table I and contrasts them with the results obtained for FACK connections. Again, $p$ denotes frequency of loss indications, and $TO$ refers to the number of timeouts per connection. There are two noteworthy aspects: First, the simplistic model $P_1$ "performs" much better for FACK connections

| $R^2$ | | Reno $P_1$ | Reno $P_2$ | Reno $P_3$ | FACK $P_1$ | FACK $P_2$ | FACK $P_3$ |
|---|---|---|---|---|---|---|---|
| all connections | | 0.74 | – | – | 0.98 | – | – |
| no loss | $(p = 0)$ | 0.99 | – | – | 0.99 | – | – |
| lossy | $(p > 0)$ | 0.62 | 0.83 | 0.92 | 0.97 | 0.91 | 0.93 |
| lossy, no timeouts | $(p > 0 \wedge TO = 0)$ | 0.99 | 0.85 | 0.85 | 0.99 | 0.91 | 0.91 |
| lossy, timeouts | $(TO > 0)$ | 0.63 | 0.86 | 0.93 | 0.97 | 0.93 | 0.93 |

TABLE III
SUMMARY OF MODELS (RENO AND FACK CONNECTIONS).

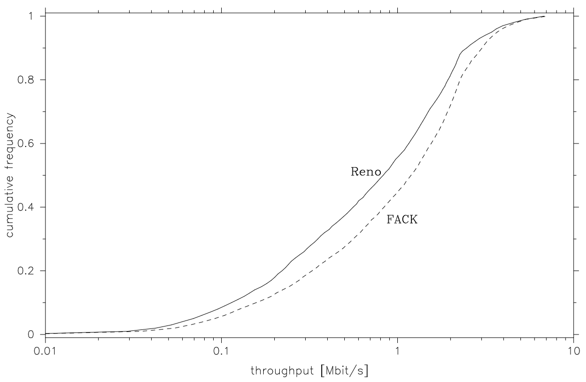


Fig. 11. Comparison of throughputs for Reno and FACK connections.

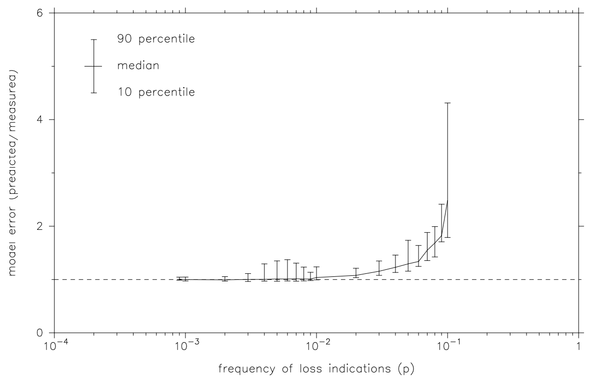


Fig. 12. $P_1$: model error as a function of loss indication frequency ($p$) for lossy FACK connections.

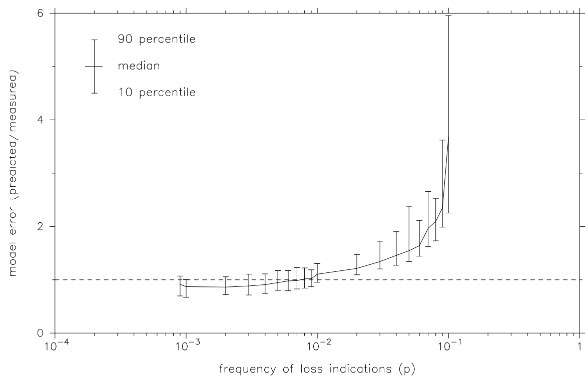


Fig. 13. $P_2$: model error as a function of loss indication frequency ($p$) for lossy FACK connections.

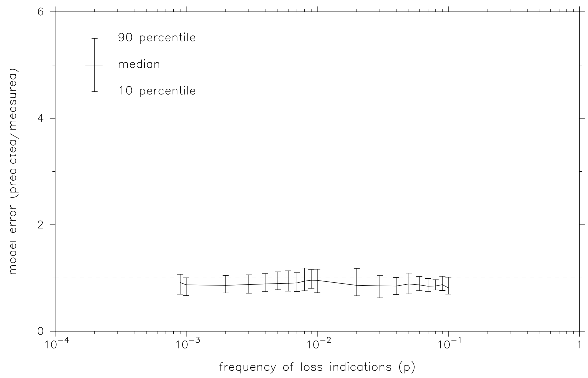


Fig. 14. $P_3$: model error as a function of loss indication frequency ($p$) for lossy FACK connections.

than for Reno connections. Second, the differences in the "quality" of the models are much less pronounced for FACK connections than they are for Reno connections. This improvement can mainly be attributed to FACK's success in avoiding retransmission timeouts.

However, caution should be applied when using a single number such as $R^2$ to characterize the "goodness" of a predictor. As we see from the numbers for $P_1$ for instance, the clusters for "lossy, no timeouts" and "lossy, timeouts" connections seem to be well described with a linear regression model, but the union of the two disjunct sets — "lossy"— is less well described with such a model. In fact, if we cluster and plot the data again according to timeouts, we see effects such as those depicted in Figure 2; however, the variation is much smaller.

This reasoning is further supported by the model error plots for lossy FACK connections shown in Figures 12, 13, and 14. Comparing these to the Figures 3, 4, and 5 allows us to see the effect of using SACK-enhanced protocols. Again $P_3$ performs best over the entire range of loss rates witnessed. In general, the clusters of the model errors appear to be tighter with fewer outliers (particularily for $P_1$). This improvement is reflected in the increase in $R^2$ as reported in Table III. However, the models also exhibit basically the same behavior for high loss probabilities (which occur mainly in connections suffering from timeouts). The reasons why (i) $R^2$ for FACK connections is higher than for Reno connections, and why (ii) $R^2$ differs only slightly for $P_1$ and $P_3$ in the case of FACK (as opposed to Reno) can be found in the distributions of the loss indication probability for the two protocols, plotted in Figure 15. The log-scaled $x$-axis represents the range of loss indication rates encountered (to be consistent with the data in Figures 12–14, we only consider lossy connections). Reno connections have a consistently higher loss indication probability than FACK connections. This property explains (i). The increase in modelling error for $P_1$ around $p \approx 0.1$ is less

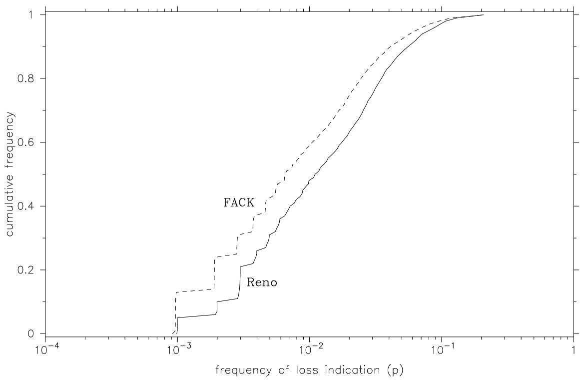


Fig. 15. Cumulative distribution of loss indication probabilities for lossy Reno and FACK connections.

pronounced in the case of FACK connections (and therefore the difference between $P_1$ and $P_3$ is smaller), because fewer FACK connections experience such loss rates than Reno connections.

The loss indication probability for FACK connections is lower than the one for Reno for two reasons: (i) the marked decrease in the number of timeouts and (ii) the fact that FACK (like other SACK enhanced protocols) treats the loss of multiple packets within one congestion window as one event (compared to standard Reno, where multiple-packet congestion signals lead to multiple recoveries and hence multiple reductions of the congestion window[7]).

*Effect on stability*. Another positive effect of using SACK-enhanced protocols can be seen in Figure 8: FACK connections experience smaller changes in bandwidth between the first and the second half of a connection than Reno connections as can be seen from the "left shift" in the CDF for the ratio of throughput change.

## IX. Concluding remarks

Adaptive applications need accurate and timely information about the currently available bandwidth. Our evaluation of three simple models to compute the bandwidth, based on a collection of detailed traces collected in the course of a 6-month Internet experiment, allows us to comment on three aspects. First, these simple estimators behave reasonably well. Although the original design goal was to obtain the asymptotic bandwidth, the estimator $P_3$ performs well enough to be considered for network-aware applications. Second, $P_3$ is based on a combination of network-level and transport-level parameters. The effectiveness of $P_3$ provides an incentive to allow an adaptive application (or the framework that is extended by the application) access to these values in lower protocol layers. Such access must be without overhead so that $P_3$ remains cheap to compute and is delivered in a timely fashion. Finally, our data also add another dimension to the results of earlier studies that reported the benefits obtained from various enhancements to TCP. These enhancements reduce the number of timeouts and thereby improve, as a side effect, the accuracy of the simple models that we investigated.

[7] This behavior is basically a bug as noted in [6] and analyzed in [5].


## Acknowledgements

We appreciate feedback and comments from B. Plattner, P. Steenkiste, H. Zhang and the anonymous reviewers.

We thank those who provided us with an opportunity to use their host(s) as a platform for our experiments: Digital (DEC SRC, Palo Alto): L. Berc; MIT (Boston): V. Strumpen; University of California (Irvine): M. Franz; University of Waterloo (Waterloo, Canada): G. Labahn; Abo Akademi (Turku, Finland): W. Weck; Universität Linz (Linz, Austria): J. Templ; Technische Universität München (Munich, Germany): K. Antreich; Universität Bern (Berne, Switzerland): G. Jäger; Univeridad de Valladolid (Valladolid, Spain): M. Revilla. D. Brennwalder contributed to the implementation of the protocols.


## References


[1] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz. TCP behavior of a busy internet server: Analysis and improvements. In *Proc. of INFOCOM '98*, pages 252–262, San Francisco, CA, March 1998.

[2] H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz. Analyzing stability in wide-area network performance. In *Proc. of ACM SIGMETRICS '97*, pages 2–13, June 1997.

[3] J. Bolliger and T. Gross. A framework-based approach to the development of network-aware applications. *IEEE Transactions on Software Engineering*, 25(5):376–390, May 1998.

[4] J. Bolliger, T. Gross, and U. Hengartner. The effectiveness of end-to-end congestion control mechanisms. Technical report, Computer Science Dept., ETH Zurich, December 1998.

[5] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *Computer Communication Review*, 26(3):5–21, July 1996.

[6] J. C. Hoe. Improving the start-up behavior of a congestion control scheme for TCP. In *Proc. of ACM SIGCOMM '96*, pages 270–280, Stanford, CA, August 1996.

[7] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM '88*, pages 314–329, August 1988.

[8] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[9] D. Lin and H. T. Kung. TCP fast recovery strategies: Analysis and improvements. In *Proc. of INFOCOM '98*, pages 263–271, San Francisco, CA, March 1998.

[10] M. Mathis and J. Mahdavi. Forward acknowledgement: Refining TCP congestion control. In *Proc. of ACM SIGCOMM '96*, pages 281–292, Stanford, CA, August 1996.

[11] M. Mathis and J. Mahdavi. TCP rate-halving with bounding parameters. http://www.psc.edu/networking/papers/FACKnotes/current., October 1996.

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018: TCP selective acknowledgment options, October 1996.

[13] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3):67–82, July 1997.

[14] R. Morris. TCP behavior with many flows. In *Proc. of ICNP '97*, Atlanta, Georgia, October 1997.

[15] B.D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz. Trace-based mobile network emulation. In *Proc. of ACM SIGCOMM '97*, pages 51–62, Cannes, France, September 1997.

[16] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proc. of ACM SIGCOMM '98*, pages 303–314, Vancouver, Canada, August 1998.

[17] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, April 1997.

[18] R. W. Stevens. *TCP/IP Illustrated: The Protocols*, volume 1. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, December 1993.

[19] W. Stevens. RFC 2001: TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms, January 1997.

[20] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proc. of INFOCOM '98*, pages 996–1003, March 1998.

[21] J. Zinky, D. Bakken, and R. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3(1):55–73, 1997.