An empirical model of TCP performance

Allen B. Downey
Olin College of Engineering
Needham, MA 02492
downey@allendowney.com

Abstract

We propose a model of TCP performance that captures the behavior of a set of network paths with diverse characteristics. The model uses more parameters than others, but we show that each feature of the model describes an effect that is important for at least some paths. We show that the model is sufficient to describe the datasets we collected with acceptable accuracy. Finally, we show that the model's parameters can be estimated using simple, application-level measurements.

1. Introduction

This paper addresses the question, "What do you need to know about a network path in order to predict the performance of TCP transfers on that path?" Our approach is model-based in the sense that we use measurements to estimate network characteristics and then use a network model to make predictions. Our approach is empirical in the sense that we try to capture all characteristics that affect performance in current networks.

A number of models have been proposed that relate TCP performance to various path characteristics, including round trip time, drop rate and bottleneck bandwidth. Most of these models focus on the steady state behavior of long transfers [9][13][14][21][15][23][22][18][6]. Because these models omit slow start, they cannot predict the performance of transfers where slow start makes up a significant part of transfer time, which are the vast majority of transfers.

Some models of short TCP transfers have been proposed [4][16][1][26][25]. These models identify two sources of variability in transfer times: variability in rtts and the effect of dropped packets. Our datasets suggest a third source of variability: nondeterminism in the growth of the congestion window during slow start. In some cases, this nondeterminism is the primary source of variability.

Another limitation of these models is that they do not address the transition from slow start to steady state (with one exception [8]). For many paths, this transition happens in the size range from 10–100 KB; it turns out that many TCP transfers fall in this range [2].

Another limitation of many previous models[1] is that they treat the drop rate as **exogenous**; that is, a characteristic of the network that is independent of the behavior of TCP. For many network paths, an important determinant of TCP performance is the occurrence of **endogenous** drops, which are caused by the transfer itself. We deal with this complexity by abstracting away the drop rate; in slow start, we estimate the probability of dropping into congestion avoidance, and in congestion avoidance, we estimate the distribution of throughput directly.

Finally, most previous models generate single-value predictions. The goal of this work is to predict the distribution of transfer times. This distribution can be used to generate predictions for whatever moments, percentiles or intervals are needed.

Thus, one contribution of this work is a model that includes all features that have a significant effect on TCP performance, including slow start, the transition to steady state, and the effect of the transfer itself on the drop rate. Of course, the price of completeness is complexity. Our model has more parameters than others, and it requires detailed measurements of a specific path. We show that this complexity is necessary; that is, a model that omits any of these features will be inaccurate for many paths in the current Internet. Conversely, we also show that the features of this model are sufficient to describe the behavior of the paths we observed (with two exceptions).

1.1. Availability of data

In order to predict the duration of a transfer, we have to know something about the network path the transfer will

[1] One exception: Misra and Ott consider the case where loss probability depends on the congestion window [17].

use. It is natural to ask what information we can reasonably expect to have.

Of course, the answer depends on the environment. In distributed environments, the performance characteristics of the resources (computers, other devices, and the network links that connect them) may be known, and traffic characteristics may be available. For example, the Network Weather Service (NWS) runs constantly in a Grid environment, monitoring the performance of a set of links and making reports available to applications [27]. The techniques we propose here could be used in this kind of environment to collect data and make predictions.

For HTTP and other TCP transfers, it is less obvious that the data needed to make a prediction are available. Users navigate the Web unpredictably, making transfers from many servers along many network paths.

To investigate the feasibility of predicting HTTP transfers, we looked at logs from the Internet Traffic Archive (`http://ita.ee.lbl.gov/`).

The BU-Web-Client dataset contains logs from WWW proxy servers at Boston University from 1998. Although users access many different servers, a small number of servers handle a significant part of the traffic; for example, the top ten servers handled 30% of the requests. This observation suggests that **a new request is likely to access a server that has been accessed many times in the past**.

To quantify this observation, we ran through the trace sequentially and, for each request, counted the number of times the same server had been accessed prior to the request. For the majority of accesses (63%), the history of previous accesses includes at least 30 requests. For a larger majority (81%) the history includes at least 10 requests.

Traces from other environments show similar patterns. The LBL-CONN-7 dataset contains traces of more than 700,000 TCP connections to and from Lawrence Berkeley Laboratories, recorded over 30 days in 1993. These connections access 12,657 unique IP addresses, but again the top ten addresses handled 32% of the connections. For each connection we counted the number of prior connections to the same address. A large majority of connections (81%) enjoy a history that includes at least 30 connections.

Assuming, then, that the historical information we need is available, how much space would be necessary to maintain it? Based on the LBL dataset, we can imagine keeping data about 20,000 remote hosts. If we keep data from the most recent 100 connections per host, and if most connections transfer fewer than 100,000 bytes, such a history would require about 32 KB per host, or a total of 625 MB.

We conclude that it is feasible to maintain a database with traces of previous TCP connections, and that for most of the paths a client uses, the database would contain information about a useful number of previous connections (10–30).

1.2. Outline

We start with exploratory measurements intended to identify the network characteristics with the biggest effect on TCP performance and, conversely, details that can be abstracted or omitted. In Section 2 we present our measurement infrastructure and the results that led us to our model. Section 3 presents the model and Section 4 the experiments that validate it.

2. Measurement

To develop our model, we wanted to collect datasets from network paths with a variety of characteristics. The ubiquity and accessibility of Web servers makes them a convenient tool for network measurements. We used GNU's `wget` (available from `gnu.org/software/wget`) to transfer large files from a variety of Web servers. We modified `wget` so that, for each transfer, it records two vectors: t_i is the time when the ith read started, and s_i is the total number of bytes read when the ith read completed.

Two of our datasets come from servers provided by collaborators, so the characteristics of the servers and parts of the network paths are known. Another 11 datasets come from servers we found in traces from the IRCache Project (`ircache.net`). Looking at one day of traces from 10 proxy servers, we identified 11 frequently-accessed files that were at least 100,000 bytes.

Each dataset includes 100 transfers with an average of 100 seconds between them (exponentially distributed). Thus, the duration of the measurements is 3–4 hours. This time scale is appropriate for our intended applications, where we expect historic information to be available, but not necessarily recent.

The characteristics of the paths we measured are diverse. Geographical locations include New York, Chicago, Colorado, California, Maine, and China. The range of path lengths is from 12 to 29 hops. The range of rtts is from 7 ms to 270 ms. The range of bottleneck bandwidths is from 350 Kbps to 100 Mbps. The range of bandwidth-delay products is from 1 to almost 2000 packets. We believe that this dataset is representative of many paths in the current Internet.

2.1. Application-level measurement

There are three general approaches to network measurement. One is to collect packet-level information somewhere in a network path. Another is to collect kernel-level information at either the sender or receiver. The third is to collect information at the application level. The packet and kernel-level approaches provide the most detailed in-

formation. Application-level measurements are easy to implement, and the resulting tools are portable.

We use application-level measurement to develop and evaluate our model. The analysis we use to estimate the model's parameters and make predictions is also applicable to passive, network-level observation of TCP transfers, but we find that the information we need is available at the application level, or can be inferred. Furthermore, in cases where timing inaccuracy is introduced, for example by context switches, these errors can be filtered out. Therefore, we expect the improvement from kernel- or network-level measurements to be small.

2.2. Path characteristics

Initially, we expect TCP performance to depend on the following path characteristics:

- Distribution of *rtt*, and correlation structure.
- Initial and subsequent congestion windows.
- Bottleneck bandwidth.
- Frequency and performance impact of drops.
- Steady-state throughput.

Some of these, like the initial congestion window, are easy to measure; others, like bottleneck bandwidth, are hard. Some have a direct effect on TCP performance; others are more indirect. Some may need to be measured explicitly; some can be abstracted away. The goal of this section is to figure out which are which.

The following sections present the steps we used to estimate path characteristics. We start by estimating bottleneck bandwidth, but we don't use use this value directly in the model. Rather, we use it to identify the breaks between flights of packets. These breaks make it possible to infer the growth of the congestion window during slow start and to estimate base rtt (round trip time without queue delays). Finally, we observe the transition from slow start to steady state and estimate steady state throughput.

2.3. Bottleneck bandwidth

The idea of using packet spacing to estimate bottleneck bandwidth was proposed by Keshav [11] and has been implemented in various network measurement tools [3][5][24][12][7].

To implement packet-pair bandwidth estimation with our measurements, we compute the first difference of the vectors t and s, yielding dt, which is the interpacket spacing, and ds, the packet sizes. For each packet, we compute the instantaneous bandwidth $bw_i = ds_i/dt_i$. If packets

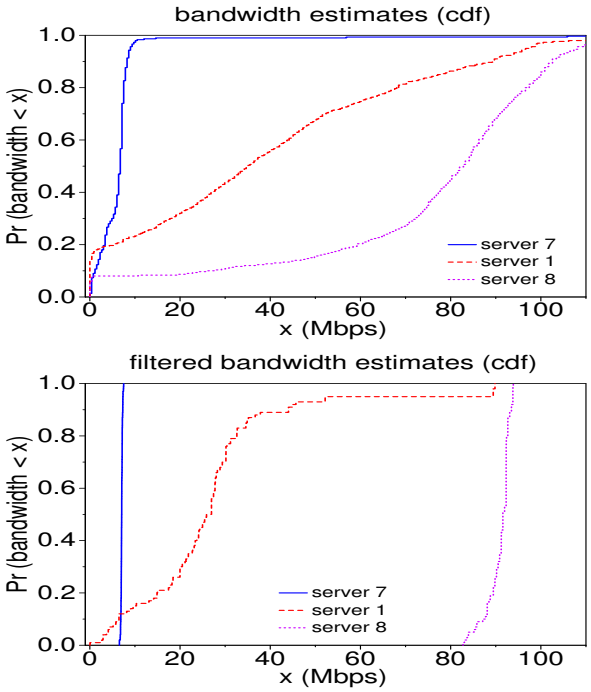


Figure 1. Bandwidth estimates before and after filtering.

leave the bottleneck link back to back, and their spacing isn't perturbed by cross traffic, *bw* estimates the bottleneck bandwidth of the path.

Current bandwidth estimators are based on the assumption that packets often arrive at the destination with unperturbed packet spacing, and that the correct bottleneck bandwidth is the mode of the distribution of estimates. Dovrolis et al. warn that under some traffic conditions, the global mode is determined by cross traffic and not bottleneck bandwidth, but they still expect a local mode at the correct value [7].

For continuous distributions, the notion of a mode is awkward to define, and methods for identifying modes tend to be ad hoc. Furthermore, many of our datasets exhibit no clear modes. Figure 1 shows distributions of *bw* estimates from servers with low, medium and high bottleneck bandwidths. Server 7 shows a strong mode around 7 Mbps, but the other two cases are less promising. Server 8 shows a mode near 90 Mbps, but the distribution is nearly uniform from 70 to 100 Mbps. Similarly, the "mode" for Server 1 spans the range from 0 to 100 Mbps. It is difficult to generate a meaningful bandwidth estimate from these distributions.

It helps that we have several observations of each path. Figure 2 plots 30 transfers of a file from a Web server, showing s versus t. Interpacket spacing is highly variable, but there are many parallel linear segments that indicate a

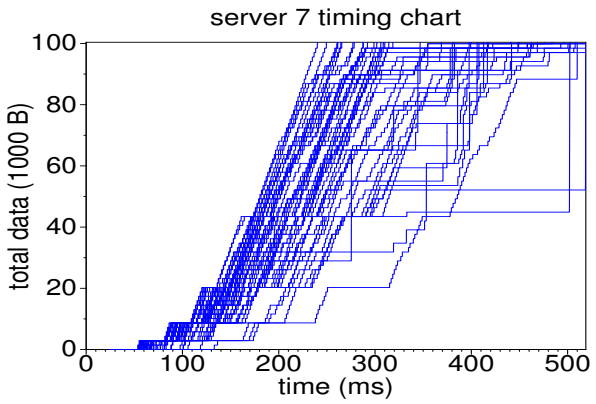


Figure 2. Timing charts for 30 HTTP transfers.

common slope. We assume that this characteristic slope corresponds to the bottleneck bandwidth, and try to estimate it statistically.

Again, we compute the vector of bandwidth estimates bw_i. We divide the vector into overlapping subsequences with length k, and compute a measure of variability for each subsequence. The variability of the the jth subsequence is

$$\sigma_j = \frac{1}{k} \sum_{i=j}^{j+k} |bw_i - m| \quad (1)$$

where m is the median of the estimates in the subsequence. The subsequences with the lowest deviation correspond to the straightest line segments in the timing chart.

We use the variability of the subsequences to filter the bandwidth estimates. We keep only the n subsequences with the lowest variability and filter the others, on the assumption that the packets were not sent back-to-back, or their interpacket spacing has been perturbed. Figure 1 (bottom) shows distributions of the estimates that remain after filtering with $k = 8$ and $n = 100$. In all three cases, the range of the estimates has been greatly reduced. Furthermore, in all of our datasets, the mode of the distribution is at or near the median; thus, we use the median as our bandwidth estimate and the interquartile distance as an indicator of its precision. This filtering technique works well with a range of values for k and n, and the estimates are insensitive to these parameters.

We can't evaluate the *accuracy* of the estimates, because we don't know the actual bottleneck bandwidths of the paths, but by dividing our datasets into subsets, we can evaluate *repeatability*. For each dataset, we generate 5 subsets with 20 randomly-chosen timing charts in each. We generate a bandwidth estimate for each subset, and compute the range and interquartile distance of the five estimates. Table 1 shows the results. The column labeled "Est bw" is the estimate based on all 100 timing charts; "Range" contains the highest and lowest estimates from each subset

Server	Est bw	Range	Interquart
1	24.908	(24.908, 28.014)	1.147%
2	63.656	(63.656, 86.860)	0.701%
3	89.040	(88.996, 90.051)	0.513%
4	92.710	(91.975, 92.712)	0.269%
5	90.677	(89.628, 91.937)	0.687%
6	63.870	(41.590, 84.870)	11.974%
7	6.982	(6.914, 7.075)	0.212%
8	91.261	(91.261, 92.313)	0.190%
9	0.331	(0.331, 0.513)	5.845%
10	9.376	(9.356, 9.412)	0.126%
11	89.474	(89.444, 90.866)	0.669%
12	22.694	(21.129, 33.811)	15.300%
13	88.775	(88.775, 90.142)	0.386%

Table 1. Bandwidth estimates.

of 20 charts; "Interquart" is one-half the distance between the 25th and 75th percentiles, written as a percentage of the estimated bw.

In most cases, the range of estimates is small, which suggests that they are actually measuring the capacity of a link in the path. However, one weakness of this technique is that it might be fooled by what Dovrolis et al. call a "post-narrow capacity mode."

2.4. Congestion windows

The duration of short TCP transfers tends to be a multiple of the round trip time, where the multiplier depends on the behavior of the congestion window at the sender. Thus, in order to predict TCP performance for a given server, we have to measure its initial and subsequent congestion windows. To do that, we have to be able to identify the end of each round of packets. The heuristics we use are similar to those implemented in T-RAT [28]; one difference is that in our datasets we are able to combine data from multiple observations of the same path.

To separate arriving packets into rounds, we look at the vector of interpacket spacing, ds, and identify intervals that seem to be due to flow and congestion control rather than queue delays. To do that, it helps to know the base rtt and the interpacket spacing at the bottleneck. As a coarse estimate of rtt, we collect the measured rtts of the SYN-ACK and request-reply rounds and compute their 5th percentile. To get the interpacket spacing at the bottleneck, we use the bandwidth estimation technique in the previous section and compute, $inter = ds_i/bw$. Then we compute a logarithmic transformation of the interarrival time, $dt'_i = f(dt_i)$, scaled so that $dt'_i = 0$ if $dt_i = inter$ and $dt'_i = 1$ if $dt_i = rtt$. This transformation gives us a criterion for breaking a timing chart into rounds; if $dt'_i > 0.5$, we consider the ith packet to be the beginning of a new round. During slow start, the breaks between rounds are obvious and the choice of this

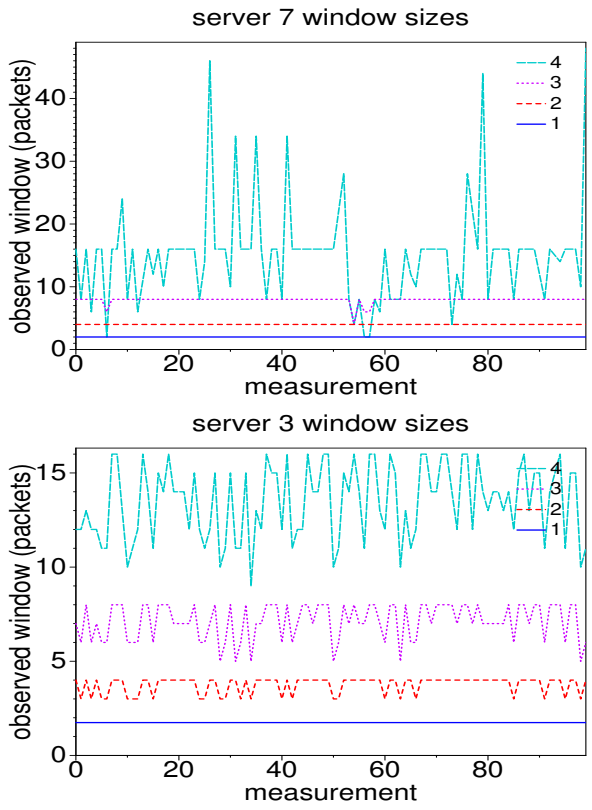


Figure 3. Window sizes for the first four rounds of slow start.

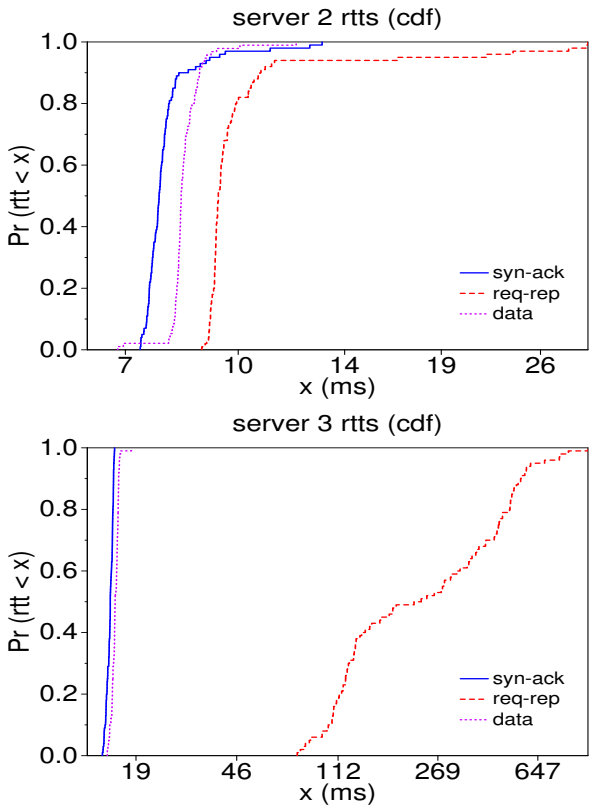


Figure 4. Distributions of round trip times for SYN-ACK, reply-request, and data rounds.

threshold has little effect. As the congestion window approaches the bandwidth-delay product, bdp, it becomes increasingly difficult to identify rounds, but at that point we stop trying to identify rounds and start trying to characterize steady-state behavior (Section 3).

Figure 3 shows estimated congestion windows for two servers. The x axis enumerates 100 transfers; the lines show the observed window, in packets, for the first 4 rounds of slow start. The top graph shows a server with the kind of slow start behavior we expect. The first round is always 2 packets, the second is always 4, and the third is usually 8, except in a few cases where, it seems, a drop causes the sender to switch to congestion avoidance. By the fourth round, the congestion window has reached bdp, which is about 15 packets, and it is no longer possible to identify the breaks between rounds accurately.

Surprisingly, this behavior is not typical. In most of our datasets, the behavior of the congestion windows turns out to be nondeterministic. The bottom graph shows an example. The initial congestion window is consistently 2524 B, a little less than 2 packets. But the second round is sometimes 3 and sometimes 4 packets. The third round is usually twice the second, but again, it sometimes falls short by a packet or two. The same thing happens in the next round; the congestion window either doubles or nearly doubles, seemingly at random. Of our 13 datasets, 10 show significant nondeterminism starting in the second or third round and continuing in subsequent rounds.

The most likely explanation of this behavior is an interaction between the delayed ACK mechanism at the receiver and the growth of the congestion window at the sender [20]. The consequence of this behavior is that variation in the growth of the congestion window is a significant source of variation in transfer time for short transfers.

2.5. Round trip time

For short transfers, the distribution of transfer times depends on the distribution of rtts. A TCP transfer takes at least two rtts, one for the SYN-ACK round and one for the request-reply round. In our datasets, we can measure the rtt of the SYN-ACK and request-reply rounds directly; after segmenting the timing chart we can estimate the rtts of the next three data rounds reliably.

Figure 4 shows distributions of *rtt* for two servers. The SYN-ACK round sees the shortest rtts because the packet sizes are minimal and there is no application-level processing at the server. The request-reply round takes the longest,

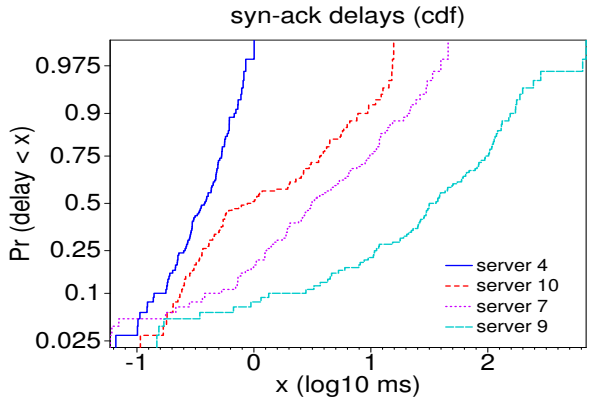


Figure 5. Distributions of delays for the SYN-ACK round, for four servers. The axes are transformed so that a lognormal distribution appears as a straight line.

and has the highest variability, because the application-level processing at the server is synchronous and may require disk access. In two of our datasets, the rtts for the request-reply round are 2–10 times longer than the network rtt. Figure 4 (bottom) shows an example. Clearly for this kind of application, a model of TCP performance needs to include a model of application-level performance.

In many cases the distribution of rtts it is well-described by a three-parameter lognormal model. For a set of rtt_i, we estimate the minimal value, $\theta = \min rtt_i$, and then compute the delays, $delay_i = rtt_i - \theta$. To see whether the lognormal model is appropriate, we plot the delays on axes transformed so that a lognormal distribution appears as a straight line. Figure 5 shows examples from four servers with a range of variability. The distributions are only approximately straight, but they are close enough that we think the lognormal parameters summarize them well.

Table 2 shows the estimated parameters for the SYN-ACK round for each server. The parameters ζ and σ are the mean and standard deviation of $\log_{10} delay$. The expected value, $E[delay]$ is $\mathrm{pow}(10, (\zeta + \sigma^2/2))$.

2.6. Correlations

The duration of a short TCP transfer is the sum of a series of consecutive rtts. Therefore, correlation between successive rtts affects the distribution of transfer times.

Bolot characterized the relationship between the rtts of successive packets and found that correlations diminish as the timescale increases, and disappear when the interval between packets exceeds 500 ms [3]. Moon et al. estimate the autocorrelation function for series of RTP packets and find strong correlations that diminish over larger intervals, again becoming insignificant beyond 500 ms [19].

Server	θ ms	ζ log_{10} ms	σ log_{10} ms	E[delay] ms
1	271.6	1.198	0.545	3.842
2	7.351	-0.379	0.424	0.749
3	14.25	-0.114	0.292	0.931
4	75.24	-0.547	0.316	0.608
5	6.648	-0.253	0.126	0.783
6	6.611	0.087	0.746	1.441
7	24.47	0.505	0.703	2.120
8	87.69	-0.414	0.598	0.790
9	37.59	1.452	0.833	6.038
10	50.94	0.009	0.662	1.256
11	227.4	1.218	0.870	4.937
12	239.6	1.960	0.238	7.303
13	50.22	-0.541	0.301	0.609

Table 2. Estimated parameters for rtts.

Server	syn-ack req-rep	req-rep data 1	data 1 data 2	data 2 data 3
1	0.203	0.180	0.220	0.364
2	(-0.113)	(0.126)	(0.076)	(-0.108)
3	(-0.082)	(-0.131)	0.702	0.770
4	(0.096)	0.265	(-0.034)	(-0.008)
5	(0.116)	(-0.121)	(0.088)	(0.092)
6	0.671	0.640	0.736	0.713
7	0.396	0.334	0.310	0.326
8	(0.044)	(-0.020)	(-0.185)	(-0.269)
9	0.721	0.682	0.657	0.638
10	0.184	(0.081)	(0.157)	(-0.144)
11	0.811	0.745	0.874	0.915
12	0.631	0.625	0.875	0.895
13	(-0.040)	(0.101)	(-0.047)	(-0.043)

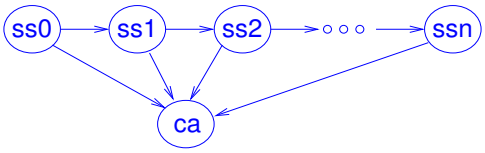
Table 3. Correlations in rtt.

By breaking our observations of slow start into a series of rtts, we can use our measurements to estimate correlations between successive rounds. Table 3 shows estimated rank correlations for the first four rounds, for each server. Values in parentheses are statistically insignificant at 90% confidence. About half of the servers show significant correlations, some larger than 0.8. All significant correlations are positive, and usually consistent from round to round.

Based on prior work, we might expect higher correlations on paths with shorter rtts, but that is not the case. There is no apparent relationship between *rtt* and the degree of correlation. On the other hand, paths with high expected delays (see $E[delay]$ in Table 2) tend to have high correlations. This result makes sense, since paths with longer delays are more likely to have queues that persist long enough to induce correlations on the relevant time scale.

3. Model

Finally we are ready to assemble a model of TCP performance. The model is based on the following state transition diagram:



The states labeled ss0 through ssn are slow start states; the state labeled ca represents congestion avoidance. For a given transfer, we can infer a sequence of states by computing observed window sizes, w_i (see Section 2.4). All transfers start in ss0. For each round, we compute the ratio of successive window sizes, w_i/w_{i-1}. If this ratio is between 1.5 and 2.0, we move to the next slow start state. If it falls short of 1.5, we assume that a dropped packet caused the congestion window to shrink and we move to ca.

In state ca we keep track of the distribution of throughputs, computed as the average throughput between the end of the last round of slow start and the end of the timing chart. Thus, for each timing chart, we compute a series of states that starts in ss0 and ends when the transfer ends or reaches congestion avoidance. We estimate the probability of each state transition by counting the number of times each transition occurs in the dataset.

3.1. Estimating transfer times

In previous sections, we have shown how to use a set of timing charts to estimate the parameters of a network path. These parameters are:

- The distribution of rtts for the SYN-ACK round, the request-reply round, and the first data round.
- The correlations between successive rounds of rtts.
- The state transition probabilities.
- The distribution of window sizes for each slow start state.
- The distribution of throughputs in steady state.

These parameters are sufficient to estimate the distribution of transfer times for a given transfer size. Here is the algorithm:

1. Set s_{total}, the total data received, to 0. Choose rtt_0 from the distribution of SYN-ACK rtts and rtt_1 from the distribution of request-reply rtts, with appropriate correlation. Set t_{total}, the total elapsed time, to $rtt_0 + rtt_1$. Start in state $ss0$.
2. Using the state transition probabilities, choose the next state at random.
3. If the new state is ca, choose $throughput$ at random from the distribution of throughputs. Compute the remaining time $t_{rem} = (s - s_{total})/throughput$ and return the sum $t_{rem} + t_{total}$.
4. Choose a window size, win, from the distribution of window sizes for the current state. If $s_{total} + win > s$, the transfer completes during this round. Return t_{total}.
5. Update $s_{total} = s_{total} + win$. Choose rtt_i from the distribution of data rtts (with appropriate correlation) and update $t_{total} = t_{total} + rtt_i$.
6. Go to step 2.

By repeating this process, we estimate the distribution of transfer times.

4. Validation

To test the model, we divide each dataset randomly into two sets of 50 transfers. We use the first subset to estimate parameters and generate a distribution of transfer times for a range of sizes. Then we compare the predicted times with the times from the second subset.

Figure 6 shows the results for four servers we chose to represent a variety of path characteristics. For Server 1, the distribution of transfer times is multi-modal (for some sizes) because the characteristics of the path changed during the measurement. For Server 2, the transfer time is determined by the distribution of rtts and the distributions are multi-modal because the growth of the congestion window is nondeterministic. For Server 9, many transfers enter congestion avoidance almost immediately; the range of transfer times is unusually wide, but the model describes the distributions reasonably well. For Server 10, performance is generally consistent and the range of transfer times is relatively narrow. In each of these cases, the model includes the features necessary to describe the distribution of transfer times, including the location and probability density of multiple modes.

In two cases, the model is less accurate. On Server 3 there is an application-level delay after the first 40 packets that is not included in the model. Server 6, it turns out, is actually three servers with different path characteristics. Subsequent requests for the same URL are handled by different servers, due to changes in DNS information caused by distributed content delivery mechanisms like those used by Akamai Technologies and Speedera Networks. Although the shape of the predicted distribution doesn't match the measured distribution, its location and variability are correct.

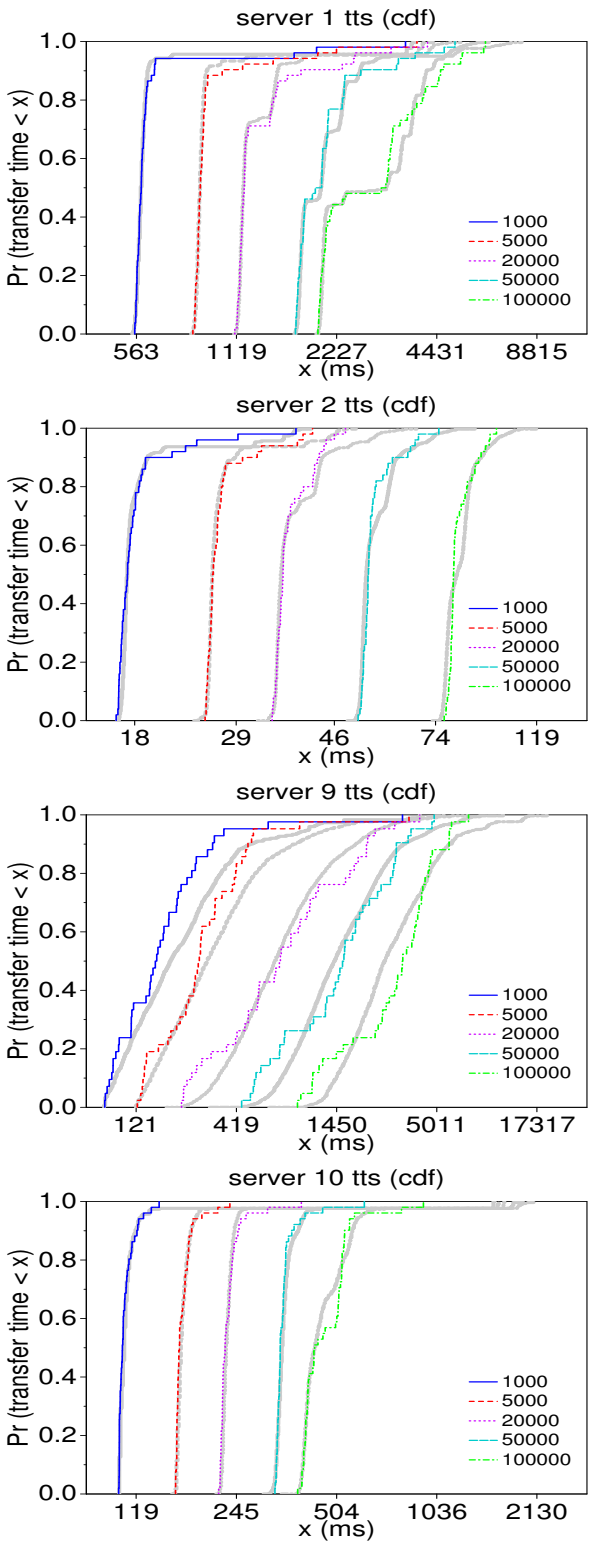


Figure 6. Distributions of predicted (thick, gray lines) and actual transfer times (thin, darker lines).

Server	Error (untruncated)	Error (truncated)
1	0.804	1.044
2	0.771	3.475
3	3.490	8.716
4	0.087	0.303
5	2.639	1.871
6	3.682	6.802
7	0.937	0.627
8	0.265	0.459
9	1.938	2.194
10	1.194	2.023
11	1.237	2.657
12	0.782	0.803
13	0.697	2.254

Table 4. Error metrics for predictions based on truncated and untruncated data.

We conclude that the model's features are sufficient to describe the behavior of the paths in our datasets, with two exceptions where the predictions are less accurate. These exceptions remind us of a fundamental limitation of model-based approaches to performance prediction: the complexity of the real world.

4.1. Extrapolating from short transfers

This limitation raises the question whether it would be better to use historical information directly rather than to estimate the parameters of the model and then use the model to generate predictions. If nothing else, this approach would be simpler. He et al. address this question, and conclude that history-based methods are as good, and under some conditions better than, model-based methods [10]. But an advantage of model-based methods is that they can make predictions about transfer sizes that have not occurred in the past. In particular, they can use measurements of short transfers to predict the duration of long transfers.

To test this capability, we use datasets truncated after the first 50,000 bytes to estimate the parameters of our model, then we use the model to predict the duration of 100,000 byte transfers. Again, we divide the datasets in half, using 50 observations to build the model and the other 50 to test it. To measure prediction error, we use the normalized area between the predicted and actual distributions, computed by summing over percentile p:

$$\sum_{p=1}^{99} \frac{|F_p^{-1}(p) - F_a^{-1}(p)|}{F_a^{-1}(p)}$$

where F_p and F_a are the actual and predicted distribution functions. This error metric doesn't mean much in absolute

terms; we use it here to compare predictions made with the truncated datasets to predictions made with the complete datasets.

Table 4 shows the results. For most servers, the predictions based on censored data are either a little worse or, by chance, a little better than the predictions that use all the data. When the predictions fail, it is usually because the truncated datasets don't show the transition from slow start to steady state. We conclude that our model can be used to predict the performance of long transfers as long as we have observed at least a few observations that are long enough to leave slow start.

5. Conclusions

Previous models of TCP performance have focused on either short transfers or steady state behavior, and most produce single-value predictions rather than intervals or distributions. The goal of this project is to evaluate the feasibility of a model-based approach to predicting the distribution of transfer times for the entire range of transfer sizes, including the transition from slow start to steady state.

Some of our conclusions are bad news: TCP performance depends on many network characteristics, some of which are hard to measure. A model that captures all of these characteristics has many parameters, which means that it requires a significant history of observations.

On the positive side, we show that the most important parameters can be estimated with simple, application-level measurements, or with passive measurements, and we argue that in our target environments it is reasonable to expect the information we need to be available.

We have developed a model that tries to capture the parameters that have the strongest effect on performance while abstracting away characteristics that have weaker effects. Features of this model include:

- It describes both slow start and steady-state behavior, so it applies to a wide range of transfer sizes.
- Rather than estimate drop rates explicitly, it incorporates both exogenous and endogenous drops in an array of state transition probabilities.
- It includes correlations between successive rtts, which is important on paths where the expected queue delays are large compared to rtt.
- It includes nondeterminism in the growth of the congestion window, which has a strong effect on the performance of many of the paths we observed.
- It abstracts away implementation details, so it is applicable to all current and most conceivable variants of TCP.

We validate this model with measurements from a small but diverse set of network paths. We show that the features included in the model are sufficient to capture the location and shape of the distribution of transfer times, even in cases where the distribution is multi-modal. Also, we show that the model is able to extrapolate from observations of short transfers to predict the performance of long transfers, which is an advantage over history-based approaches.

5.1. Limitations

Application-level measurements are easy to implement, and tools that use them are portable. The price of this convenience is that some of the things we would like to measure, like window sizes, are not directly visible to an application. We have shown that it is possible to infer this information with acceptable accuracy, but there are two parts of the model that would benefit from network-level information.

The first is identifying dropped packets. The heuristics our model uses are successful in the sense that they identify characteristics in a timing chart that indicate a dropped packet, but without network-level traces we can't assess their accuracy.

The second limitation is the difficulty of distinguishing server delays from network delays. For the HTTP transfers we looked at, most server delays occur during the request-reply round; after that, the servers kept up with the network. Other kinds of TCP transfers, like ftp, may be similar, but there are other cases where a more detailed model of server performance may be necessary.

Finally, an aspect of TCP performance that we left out of the model is the effect of dropped packets at the end of a transfer. In our datasets, most drops were caught by the Fast Retransmit mechanism, so they tended not to impose long delays, except indirectly by reducing the congestion window. When a packet is dropped at the end of a transfer, there may not be enough ACK packets to trigger Fast Retransmit, and a transfer may suffer a timeout. In our datasets, these events are rare, but their effect is significant.

References

[1] C. Barakat and E. Altman. Performance of short TCP transfers. In *NETWORKING*, volume 1815 of *Lecture Notes in Computer Science*, pages 567–579. Springer, 2000.

[2] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns: Characteristics and caching implications. *World Wide Web, Special Issue on Characterization and Performance Evaluation*, 2:15–28, 1999.

[3] J.-C. Bolot. Characterizing end-to-end packet delay and loss in the internet. *Journal of High Speed Networks*, 2(3):289–298, September 1993.

[4] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *INFOCOM (3)*, pages 1742–1751, 2000.

[5] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical Report TR-1996-006, Computer Science Dept., Boston University, 1996.

[6] C. Casetti and M. Meo. A new approach to model the stationary behavior of TCP connections. In *INFOCOM (1)*, pages 367–375, 2000.

[7] C. Dovrolis, P. Ramanathan, and D. Moore. Packet dispersion techniques and capacity estimation. *IEEE/ACM Transactions on Networking*, December 2004.

[8] N. Ehsan and M. Liu. Analysis of TCP transient behavior and its effect on file transfer latency. In *IEEE ICC*, May 2003.

[9] S. Floyd. Connections with multiple congested gateways in packet-switched networks, Part 1: One-way traffic. *ACM Computer Communication Review*, 21(5):30–47, Oct. 1991.

[10] Q. He, C. Dovrolis, and M. Ammar. Prediction of TCP throughput: Model-based and history-based methods. In *Sigmetrics*, 2005.

[11] S. Keshav. A control-theoretic approach to flow control. In *SIGCOMM*, pages 3–15, 1991.

[12] K. Lai and M. Baker. Measuring bandwidth. In *IEEE INFOCOM*, pages 235–245, 1999.

[13] T. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, July 1997.

[14] T. V. Lakshman, U. Madhow, and B. Suter. Window-based error recovery and flow control with a slow acknowledgement channel: A study of TCP/IP performance. In *INFOCOM (3)*, pages 1199–1209, 1997.

[15] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), 1997.

[16] M. Mellia, I. Stoica, and H. Zhang. TCP model for short lived flows. *IEEE Communications Letters*, 6(2):85–88, February 2002.

[17] A. Misra and T. J. Ott. The window distribution of idealized TCP congestion avoidance with variable packet loss. In *INFOCOM (3)*, pages 1564–1572, 1999.

[18] V. Misra, W. Gong, and D. Towsley. Stochastic differential equation modeling and analysis of TCP windowsize behavior. Technical Report ECE-TR-CCS-99-10-01, University of Massachusetts, 1999.

[19] S. B. Moon, J. Kurose, P. Skelly, and D. Towsley. Correlation of packet delay and loss in the Internet. Technical Report UM-CS-1998-011, University of Massachusetts, March 1998.

[20] W. Noureddine and F. Tobagi. The transmission control protocol. Technical report, Stanford University, July 2002.

[21] T. Ott, J. Kemperman, and M. Mathis. Window size behavior in TCP/IP with constant loss probability. In *IEEE HPCS*, June 1997.

[22] J. Padhye, V. Firoiu, and D. Towsley. A stochastic model of TCP Reno congestion avoidance and control. Technical Report CMPSCI 99-02, University of Massachusetts, 1999.

[23] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP throughput: A simple model and its empirical validation. In *SIGCOMM*, pages 303–314, 1998.

[24] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[25] B. Sikdar, S. Kalyanaraman, and K. Vastola. Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno and SACK. In *IEEE GLOBECOM*, pages 1781–1787, November 2001.

[26] B. Sikdar, S. Kalyanaraman, and K. Vastola. TCP Reno with random losses: Latency, throughput and sensitivity analysis. In *IEEE IPCCC*, pages 188–195, April 2001.

[27] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[28] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of Internet flow rates. In *SIGCOMM*, 2002.