Significantly improving networked applications using congestion-aware transport modeling

Yi Cao, Aruna Balasubramanian, Anshul Gandhi
{yicao1, arunab, anshul}@cs.stonybrook.edu
Stony Brook University

Abstract
Given the growing significance of network performance, it is crucial to examine how to make the most of available network options and protocols. We propose ECON, a model that predicts performance of applications under different protocols and network conditions to scalably make better network choices. ECON is built on an analytical framework to predict TCP performance. Unlike existing models that make limiting assumptions about packet loss, ECON infers a relationship between loss and congestion using empirical data. The empirical relation drives an online and dynamic analytical model to predict TCP performance. ECON then builds on the TCP model to predict latency and HTTP performance. Across three wired and one wireless network, our model outperforms five alternative TCP models. We show how ECON builds on the TCP model to (i) accurately predict latency for transferring data of a given size between a client and server, (ii) easily and scalably choose between HTTP/1.1 and HTTP/2 given a workload and network condition, and (iii) determine the best data transfer path in a cloud deployment among paths with different network characteristics.

1 Introduction

The performance of most Internet and data center applications, including Web servers [5, 6], multi-tier cloud services [64, 74], and big data processing applications [27], depends critically on the underlying network. A major challenge for network practitioners is choosing the right network settings among different available options.

The problem is that the right choice is not always obvious. For example, a Web service provider must choose between multiple application layer protocols: HTTP/1.1 (with varying number of connections) versus the relatively new HTTP/2 protocol [39]. Prior work has shown that the choice between HTTP/2 and HTTP/1.1 is not straightforward [73]. Likewise, data center operators must choose between network paths that have different round trip times (RTT) and loss rate characteristics. The choice is complicated as both factors impact performance in different ways

However, empirically evaluating all possible options under different workloads and network environment conditions is infeasible. Worse, making the wrong choice can severely impact application performance, *by as much as 4.5×*, as we show in §6.

In this paper, we present ECON, an accurate model-based and measurement-based approach to systematically and scalably compare network choices. The performance of higher-layer applications depends heavily on the performance of the underlying transport layer, especially TCP [73]. Our key idea is to first analytically model the underlying TCP performance, and then use this as the building block for ECON.

Unfortunately, modeling TCP performance in today's networks is non-trivial. TCP performance is tied to network congestion, which in turn is dynamic and cannot be easily captured by analytical models. Existing TCP models [19, 22, 24, 57] assume either that congestion is a constant or that it follows a known distribution, which leads to inaccurate models (§5). To further exacerbate the problem, the congestion in the network depends on external factors, including cross traffic and intermediate router buffers, but end-points have little visibility into these factors.

ECON builds an analytical TCP model, but augments the analysis with real-time empirical measurements to account for network congestion[1]. Congestion control algorithms typically regulate TCP throughput based on the occurrence of losses [3, 35]. ECON thus uses empirical data to *infer* the relationship between loss rate and congestion window (cwnd), where cwnd is the amount of data sent in one RTT. Importantly, by only relying on observations of end-to-end performance, ECON is able to capture the effect of competing traffic *without* requiring explicit information about the cross traffic or the intermediate routers, which is hard to obtain.

The core of our TCP modeling approach is to estimate the amount of data transferred between loss events. By leveraging the cwnd evolution dictated by the underlying congestion control algorithm, and using online information to track how loss rate depends on cwnd, ECON accurately predicts the TCP throughput in the next interval. In this paper, we derive the model for TCP CUBIC [35], which is the default TCP variant on Linux and Mac OS, and also show how our model can be extended to TCP Reno [3]. Further, ECON explicitly models the *slow-start phase* of TCP, unlike most existing models [19, 22, 24, 57], and is thus able to accurately predict the performance of short flows.

ECON adapts to variations in real-world network conditions by using a sliding-window approach to update its empirical

[1]Hence the name ECON model that stands for *Empirically-Augmented Congestion Aware Network Model.*

inputs. Further, by capturing the effect of network congestion, ECON can accurately model the performance of *parallel* TCP connections.

We build on ECON's TCP model to predict the performance of HTTP/1.1 and HTTP/2. To this end, ECON extends the continuous flow model derived for TCP to *finite flows*, since HTTP applications work with finite, discrete objects. Because ECON models parallel TCP connections, it can accurately compare the performance of HTTP/1.1 (which uses parallel connections) and HTTP/2 (which multiplexes requests on a single connection). We also build on ECON's TCP model to predict the data transfer latency for any flow.

ECON is light-weight and practical. The ECON daemon at the sender periodically collects empirical data from existing flows. Given a set of network choices during runtime, ECON predicts the performance for the different choices online. Our goal is to model network performance under existing choices to empower practitioners, and not to develop new TCP or HTTP variants or to debug network performance based on offline analysis.

We rigorously evaluate ECON's TCP model with over 700 hours worth of experiments spread over several months across (1) three different wired networks, including those within and across different Azure public cloud sites, (2) a wireless network on the US East Coast, and (3) under different TCP variants (CUBIC [35] and Reno [3]). We compare ECON with five alternative models—the classical PFTK model [57], two other analytical models [22, 24], and two history-based models [36]. Our results consistently show that our modeling error is substantially lower than other models, often by 65%, even under dynamic network conditions. Our median throughput modeling error is *8%–16%* across all experiments.

For Web applications, ECON predicts the latency of the HTTP/1.1 and HTTP/2 with an average error of 3% and 7.5% respectively . Importantly, ECON accurately predicts the workload and network conditions under which HTTP/2 outperforms HTTP/1.1 and vice-versa (see §6), without requiring exhaustive experimentation, unlike prior work [73].

Finally, we show how ECON can help choose the best data transfer path in a cloud deployment, among paths with varying RTT and loss rate characteristics. We find that pairs of paths with the same ratio of RTT and loss rate can still exhibit vastly different gains in throughput, suggesting that empirically derived rules-of-thumb can lead to significant loss in performance. By contrast, ECON accurately predicts the performance for *any* connection, allowing practitioners to efficiently choose among available data transfer paths.

2 Background and Motivation

Transmission Control Protocol (TCP) is the widely used protocol in the transport layer [26]. This section provides an overview of TCP and motivates the need for a new TCP modeling framework.

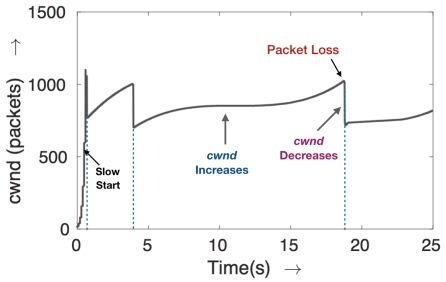


Figure 1. TCP CUBIC congestion window size (cwnd) evolution as a function of time. In the slow-start phase, cwnd doubles until the slow-start threshold is reached. After this, cwnd increases according to a cubic function in Eq. (1). Upon a loss, cwnd is reduced by a multiplicative factor.

2.1 Background on TCP

One of the key features of TCP is congestion control. If the TCP sender sends too many packets, the intermediate router buffers overflow leading to packet loss. If the TCP sender sends too few packets, the network is under-utilized. TCP regulates the amount of data it sends in one round trip time (RTT) using the *congestion window* or cwnd parameter.

TCP typically starts in the *slow-start phase* where cwnd is increased exponentially until a slow start threshold is reached or a loss occurs. Then, TCP switches to *congestion-avoidance* phase. In this phase, most TCP variations [2, 3, 35] increase cwnd less aggressively, until there is a loss. The exact cwnd regulation depends on the TCP variant in use, but the rise and fall of cwnd is periodic.

Figure 1 shows the evolution of cwnd over time (in terms of RTTs) under TCP CUBIC [35] for the *SouthEast* testbed (see §5. After slow-start, TCP increases the congestion window via the equation:

$$cwnd_{cubic} = C(t - K)^3 + W_{max}, \tag{1}$$

where C and K are constants, t is the elapsed from the previous packet loss, and W_{max} is the cwnd size just before the last packet loss. When a loss occurs (as indicated by triple duplicate acknowledgements) the cwnd drops by a multiplicative factor of $(1 - \beta)$, with $0 < \beta < 1$. If the loss occurs due to timeout, which is considerably rare, TCP goes back to slow start phase.

A popular approach is to use an Additive Increase, Multiplicative Decrease (AIMD) algorithm to regulate cwnd. The algorithm conservatively increases cwnd using an additive factor in each round. But when a loss occurs, it rapidly reduces cwnd using a multiplicative factor. For example, TCP Reno [3] increments cwnd by 1 in each RTT, and drops cwnd by a factor of 2 upon a loss.

2.2 Overview of existing analytical models

Researchers have developed analytical models to characterize TCP throughput (and/or latency) as a function of various network parameters [13, 19, 22, 24, 53, 57], starting with the classical PFTK model [57]. These existing analytical models

focus on TCP Reno [3], and all but one model focus only on the congestion avoidance phase.

2.3 Limiting assumptions of existing models

Existing models for TCP performance [19, 22, 24, 53, 57, 58] make several assumptions that are not valid in practice, including:

1. *Packet losses are independent and the loss probability is constant.* An important assumption that the existing models [13, 19, 22, 24, 53, 57] make is that loss rate can be modeled analytically, either as a constant or as some distribution that is known a priori, e.g., Poisson.
   Unfortunately, this assumption is false. Figure 2 shows the empirical relationship between loss probability and cwnd for a real-world testbed (for more details, see §3.2). The loss probability certainly depends on the congestion window size and is not a constant.
   Since loss and cwnd are assumed to be independent, a corollary is that TCP can keep adding more parallel connections (and increase cwnd) without affecting losses. Clearly this is not true, and our empirical results in Figure 2 show the effect of the number of parallel connections on loss. This incorrect assumption is especially problematic when comparing HTTP/1.1, that uses parallel TCP connections, and HTTP/2 which only uses a single TCP connection.
2. *Ignoring the starting congestion window size.* The starting congestion window size varies across flows, especially when TCP connections are reused. However, existing models assume that cwnd values just before a loss are identically distributed. While the effect of the starting congestion window is amortized for bulk transfers, the impact can be significant for short data transfers (see §5.4 and §7.2).
3. *Ignoring dynamic network conditions.* Existing analytical approaches typically focus on static network conditions where the parameter values, such as loss probability, are assumed to be stable. However, this is not true in practice [21, 36, 40, 71].

These limiting assumptions also make these models inaccurate in practice (see §5), motivating the need for a better modeling framework.

3 ECON TCP Model

This section presents the core of ECON–an accurate, congestion-aware TCP throughput model. ECON combines an analytical model with empirical data on network congestion. By doing so, ECON removes the limiting assumptions made in prior models. The TCP model is the foundation for application layer modeling that we discuss in the next section.

3.1 Model intuition

We first describe a high-level overview of the ECON TCP model. Figure 3 illustrates the evolution of cwnd under TCP CUBIC [35] during the congestion-avoidance phase. We model the throughput for this congestion-avoidance phase and then model slow-start separately (§3.5).

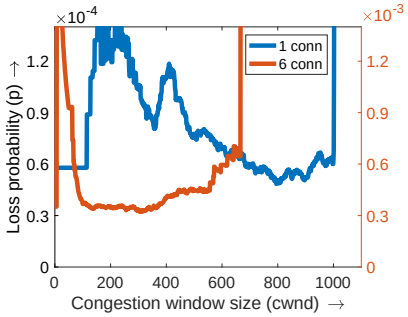


Figure 2. Empirical results for loss probability (p) vs. congestion window size (cwnd) for Northeast under TCP CUBIC. The figure shows that the relationship between loss and cwnd also depends on the number of parallel connections. The loss rate for 6 parallel connections shown in red.

ECON leverages the periodicity of the congestion window behavior at steady state to predict throughput of the current triple duplicate period. The triple duplicate period (TDP) is defined as the period between a starting congestion window and the first loss, indicated by triple duplicate acks. During this period, the congestion window rises and then falls abruptly at the end, due to a loss.

The throughput of a TDP can be estimated as the amount of data sent during the period divided by the length of the period. For example, starting from a starting congestion window scwnd in Figure 3, the throughput is a function of the number of packets sent between scwnd and W_{max} and the length of the TDP.

The key challenge is in predicting when the loss will occur. Unlike prior work, ECON does not assume that loss rate is constant. In fact, as indicated in Figure 2, the loss rate will change at each point in the TDP curve because the loss rate depends on the congestion window size. To this end, ECON takes as input (i) the empirical relationship between loss rate and cwnd, and (ii) the starting congestion window value, to estimate the expected number of packets before the loss occurs. The throughput then is a function of the number of packets sent before a loss.

Below we describe how we obtain the relationship between loss rate and cwnd, and the ECON analytical TCP model.

3.2 Empirical relation between cwnd and loss rate

To derive the relationship between loss probability, p, and cwnd, we monitor the cwnd values of existing TCP flow(s) and mark the cwnds where loss occurs. Losses can be inferred based on the drop in cwnd values and the deterministic nature of the TCP window evolution.

p(cwnd) is the loss probability at cwnd and is estimated as the number of losses recorded at cwnd divided by the total packets sent at that cwnd. Some congestion windows may not be encountered during a monitoring period. In that case,

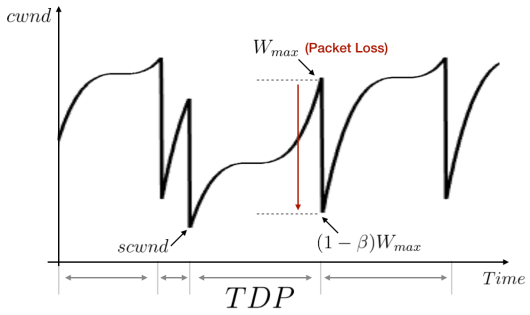


Figure 3. Illustration of cwnd evolution for CUBIC. The ECON TCP model estimates the number of packets sent before the first loss after a starting congestion window (e.g., the area under the curve from scwnd to W_{max}) and divides it by the time to first loss (marked TDP) to estimate throughput. Different from existing models, ECON does not assume that the loss rate is constant. Instead, ECON empirically determines loss rate as a function of cwnd.

we set the loss rate at the cwnd to be the average loss across all cwnds. For all congestion windows beyond the maximum encountered cwnd, we set the loss rate to 0.

Since we infer the function p(cwnd) by observing losses at the end point, we do not need to specifically model the effect of queue management techniques such as RED [16, 54].

p(cwnd) relationship for real-world networks: Figure 2 shows the empirically obtained p(cwnd) curve based on one hour of monitoring under *Northeast* under TCP CUBIC. Not only does p depend on cwnd, but the relationship depends on the number of parallel connections.

The p(cwnd) relationship is a U-shaped curve and *not* monotonic. The loss rate initially decreases with an increase in cwnd, but then the relationship is reversed due to congestion. At the left side of the curve, the cwnd drops under poor network conditions because of multiple losses. Thus, conditioned on the fact that the current cwnd is small, we can infer that the network conditions is poor and loss rate is high. We obtain a similar U-shaped curve in all the other network setups in our experiments, including those over WiFi (not shown here for brevity).

Parallel TCP connections: The p(cwnd) relationship also depends on the number of parallel connections as the bandwidth is shared between more connections. Figure 2 shows the difference in the p(cwnd) relationship for 1 and 6 parallel connections. In our experiments, we capture the empirical relationship p(cwnd, npc), where npc is the number of parallel connections by tracking the losses at different congestion window sizes across connections.

3.3 Modeling TCP CUBIC throughput

We now describe our TCP throughput modeling approach for TCP CUBIC, which is the most widely used TCP variant, being the default version on Mac OS and Linux.

ECON takes as input the empirical p(cwnd, npc) relationship and the starting congestion window scwnd. Then, the

TDP	Triple Duplicate Period
cwnd	congestion window size
scwnd	starting congestion window size
npc	# of parallel connections.
N	random variable denoting # of packets sent in a TDP
M	random variable denoting # of packets before loss is detected
X	random variable denoting # of RTTs in a TDP
$p(x)$	loss probability at congestion window size x.
W_i	size of the congestion window before i^{th} loss

Table 1. Parameters and terminology used in this paper.

throughput B is given by

$$B(scwnd, p(cwnd, npc)) = E[N]/((E[X]+1)\cdot RTT), \quad (2)$$

where N denotes the number of packets sent in a TDP and X denotes the number of RTTs until the first loss is detected. The total throughput is obtained by summing the throughputs of all parallel connections.

The goal now is to derive N and X which depends on the inputs, scwnd and p(cwnd,npc), and the cwnd evolution determined by the CUBIC congestion control algorithm.

Let M denote the index of the first lost packet; (N-M) is simply the number of packets sent after the first loss occurs but before it is detected. We first derive the expected number of packets before the first packet loss, $E[M]$ and then use it to derive the expected number of packets sent in a TDP, $E[N]$.

Deriving M, the index of the first packet loss :

The expected number of packets before the first packet loss, $E[M] = \sum_i i \cdot Pr(M = i)$, where $Pr(M = i)$ is the probability loss occurred at packet i..

When there is no loss, the cwnd increases every RTT according to Eq. (1). For now, let npc = 1 and the time corresponding to the beginning of this TDP be $t_0 = 0$.

The probability that the very first packet is lost is simply the probability that a loss occurs at the starting congestion window scwnd; thus, $Pr(M = 1) = p(scwnd)$. The probability that the first loss is at the second packet is similarly given by $Pr(M = 2) = (1 - p(scwnd)) \cdot p(scwnd)$. Thus, for any i^{th} packet sent in the first congestion window (of size scwnd), we have $Pr(M = i) = (1 - p(scwnd))^{i-1} \cdot p(scwnd)$.

If the first loss happens after the first congestion window, we have to take into account the change in loss probability, since loss depends on the congestion window size. In general, if the loss happens at the m^{th} packet of the n^{th} congestion window, we have

$$Pr(M) = \left(\prod_{j=1}^{n-1}(1 - p(cwnd_j))^{cwnd_j}\right)\cdot(1-p(cwnd_n))^{m-1}\cdot p(cwnd_n),$$

where $cwnd_j$ is the cwnd in the j^{th} RTT ($cwnd_1 = scwnd$).

By conditioning over M and generalizing to npc parallel connections, we get:

$$E[M] = \sum_{i=1}^{\infty} i \times \left(\prod_{j=1}^{i-1} (1 - p(cwnd_j, npc)) \right) \times p(cwnd_i, npc) \quad (3)$$

Deriving N, the number of packets sent in a TDP: The number of packets sent in a TDP, N, is the number of packets sent before a loss is detected (M) except the lost packet, and the additional packets sent before the loss is detected by the sender (one RTT). We thus have:

$$E[N] = (E[M] - 1) + cwnd_{E[X]}, \quad (4)$$

where E[X] is the expected number of RTTs between scwnd and the first loss. We use a similar approach to derive E[X] as we did for E[M] (see Appendix A). The intuition is that each RTT corresponds to one congestion window.

Based on the E[N] and E[X] we can obtain the throughput B according to Eq. (2). In §4.3 we describe how we estimate the bandwidth in practice.

3.4 Extension to TCP Reno

The above model can be extended to other TCP variants. We consider TCP Reno as one example. Under Reno, the cwnd increases by 1 each RTT and decreases to half the value in the event of a loss.

As before, the cwnd evolution has a repeating pattern, so we can focus on an arbitrary TDP to estimate the throughput. Our modeling approach for Reno (see Appendix A) is similar to CUBIC, except that the cwnd evolution that dictates the p(cwnd) function and the derivation of E[M] and E[N] is now based on Reno's AIMD algorithm.

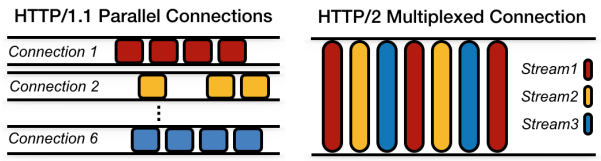
3.5 Modeling TCP Slow start

Similar to the congestion avoidance phase, the estimated throughput in the slow start phase is a function of (a) the estimated number of packets sent in slow-start before a loss occurs or the slow-start threshold is reached and (b) the evolution of cwnd during slow start. The throughput during slow start is then

$$E[B_s] = E[N_s]/((E[X_s] + 1) \times RTT) \quad (5)$$

where N_s is the number of packets sent during slow start before TCP switches to congestion avoidance phase and X_s is the length of the slow start phase.

Let the cwnd at the beginning of slow start be icwnd (10, in our setup when initiating a new connection). Then, by the cwnd evolution during slow-start, for the k^{th} RTT, $cwnd = icwnd \times 2^{k-1}$ since cwnd doubles every RTT. If there are npc parallel connections, then the loss probability for any packet in this RTT is $p(icwnd \times 2^{k-1}, npc)$. N_s, the expected number of packets before a loss is estimated using this loss probability. We use these two parameters to derive an expression for expected slow-start throughput. See Appendix for a formal derivation for the parameters in Eq. (5).



(a) HTTP/1.1 creates parallel TCP connections.

(b) HTTP/2 multiplexes over a single connection.

Figure 4. HTTP/1.1 versus HTTP/2.

4 ECON

The TCP modeling is the building block over which we build ECON's application layer prediction. ECON is a practical, online, model that can adapt to dynamic network changes.

4.1 Modeling latency

Our first step towards application-layer modeling is to model the latency of a TCP flow. Consider data of size x to be transmitted at time t on a single connection; we extend this to multiple connections in the context of HTTP next. We estimate the latency as a function of throughput. If the data transfer is part of an existing flow, then we set *scwnd* in the equations to be the congestion window size at that time, say $cwnd_t$; else, we use our slow-start model (see §3.5) to determine *cwnd*. The throughput is given by Eq. (2) and Eq. (5). Finally, the latency to transfer data of size x is estimated as $x \cdot RTT/(B \cdot MSS)$, where *MSS* is the maximum segment size.

4.2 HTTP modeling

Both HTTP/1.1 and HTTP/2 work over TCP. In both cases, the client sends an HTTP request for an object and the server sends an HTTP response with the requested object. We extend our (continuous flow) TCP model to HTTP by considering *finite flows* since HTTP applications work with finite, discrete objects. To this end, our HTTP model proceeds in epochs, where the length of each epoch is the estimated RTT. Inputs to our model are the size and number of objects being requested and, for HTTP/1.1, the number of parallel connections (*npc* in our model).

The difference between HTTP/1.1 and HTTP/2 is in how the TCP connections are leveraged. For each request, HTTP/1.1 creates a new TCP connection or reuses an existing idle connection. To improve performance, clients often create parallel TCP connections to request multiple objects simultaneously. Figure 4(a) illustrates HTTP/1.1 with parallel connections. Most Web browsers limit the number of parallel connections per server to six [75].

HTTP/2 [39], on the other hand, multiplexes multiple HTTP requests and responses in a single TCP connection rather than creating multiple connections. HTTP/2 refers to multiplexed object requests as *streams* and has a limit on the number of streams [14]. Figure 4(b) shows HTTP/2 where the payload is multiplexed over a single TCP.

HTTP/1.1 model: In the first epoch, say starting from time t_s, one object is assigned to each of the npc parallel TCP connections; for the purposes of modeling, the assignment order of objects is not important. We then use our per-connection throughput estimation from Eq. (2), along with npc and the p(cwnd) empirical relationship to predict the number of epochs needed to complete the first (or fastest) transfer. After this transfer, the next outstanding object is assigned to this connection in the *subsequent* epoch; if there are no outstanding objects, we modify the number of parallel connections and use the corresponding p(cwnd, npc) function. This process continues until all objects are transferred, say at epoch ending at time t_e. The predicted latency is then $(t_e - t_s)$.

HTTP/2 model: In the case of HTTP/2, we only have one connection. Further, we can treat all outstanding objects as one combined request that must be transferred over the one connection. We thus use our throughput model from §3.3 to predict the transfer time. Unlike HTTP/1.1, no RTTs are wasted when a new stream is added to replace a completed stream in HTTP/2.

4.3 Using the model in practice and adapting to network changes

To use the model in practice, one first empirically obtains the p(cwnd) function as described in §3.2. We can piggyback the estimation of p(cwnd) and other model parameters using existing TCP flows between a (sender, receiver) pair.

In practice, the p(cwnd) relationship can change dynamically for several reasons, such as failures or rerouting along the path and increased traffic due to colocated flows anywhere on the path [12, 43].

We use a "sliding window" approach to monitor the network conditions periodically, including RTT, losses, and the number of packets sent at each congestion window, over a certain sliding window size. We use this to dynamically update the model parameters and the p(cwnd, npc) function. We use sensitivity analysis experiments to find the best window size for the sliding window. For a fair comparison, we use a similar sliding window approach for other TCP models when applicable (see §5).

Using the empirical measurement, we drive first estimate expected throughput using Eq. (2). This estimation is closed-form except for the infinite summation over cwnd values, e.g., in Eq. (3). In practice, we rarely see cwnd values greater than a few thousands, so we cap the summation. The third step is to periodically update the model parameters and p(cwnd) function via the sliding window approach discussed in §4.3.

5 ECON TCP Model Evaluation

We evaluate our TCP throughput and latency modeling accuracy under three different real world networks for: (i) TCP CUBIC, (ii) TCP Reno, and (iii) a WiFi network. We compare our model's performance with several alternative models—three of these are formula-based (FB) models [22, 24, 57] and the other two are history-based (HB) models [36].

5.1 Methodology

Networks and experimental setup

We conduct experiments on three real-world networks (we omit exact locations to preserve author anonymity):

1. **Azure:** This is a collection of five networks, with the sender VM in each case hosted in the East US location of Azure public cloud and the receiver VM hosted in the Japan East, East US 2, West US 2, Central US, and South Central US locations of Azure public cloud [51]. The average RTT within *Azure* is 10–200ms depending on the receiver VM location. Unless specified otherwise, we report results for the receiver VM in Japan East, with average RTT of 200ms.
2. **Southeast:** The receiver VM is located in a Southeast US site of CloudLab and the sender is located in Northeast US with average RTT of 23ms. The sender-receiver RTT is quite small for this network, resulting in constant cwnd (due to insufficient data generation to fill the cwnd in each RTT). We thus use TC [1] to add 20ms to RTT.
3. **Northeast:** The sender and the receiver are both in Northeast US and are connected by a switch with an average RTT of 50ms. We use TC to add 50ms to RTT and set a packet loss rate of 0.001%.

The p(cwnd) curve for *Northeast* network under TCP CUBIC and for *Azure* network under TCP Reno is shown in Figure 2; graphs are similar for other networks. When running our experiments, we leave the network parameters in the default state. For example, TCP SACK and delayed ACK are enabled. We find that the modeling error is not affected much by the SACK and Delayed ACK settings. The one parameter that we do change is the Linux TCP receive buffer size, which we set to the maximum allowable value under the OS, (2^{31}-1) bytes)); this is done so that the data transfer is not limited by small receive buffer sizes.

Data collection

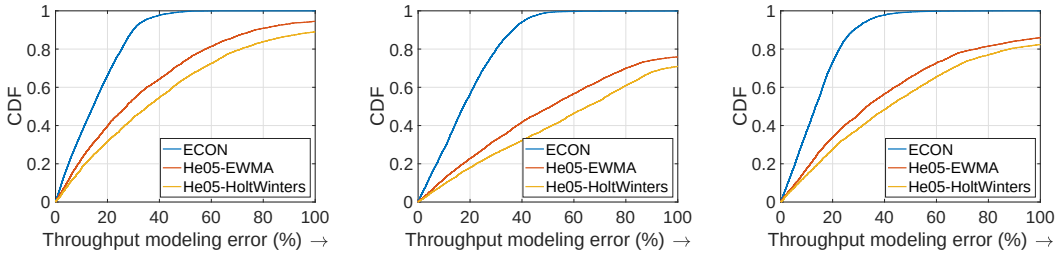
All senders and receivers run the Ubuntu OS (v14.04.5 or later). We use *iPerf* [68] to send TCP traffic from sender to the receiver. For each network, we run several 1-hour experiments spread over a period of 12 months with varying number of TCP connections (from 1 to 10); in total, we have about 700 hours worth of experimental data. Each experiment begins with the slow-start phase.

We use the Linux TCP probe module [4] to record the congestion window size (cwnd) when sending packets and estimate the p(cwnd, npc) function as described in §3.2.

Alternative models for comparison

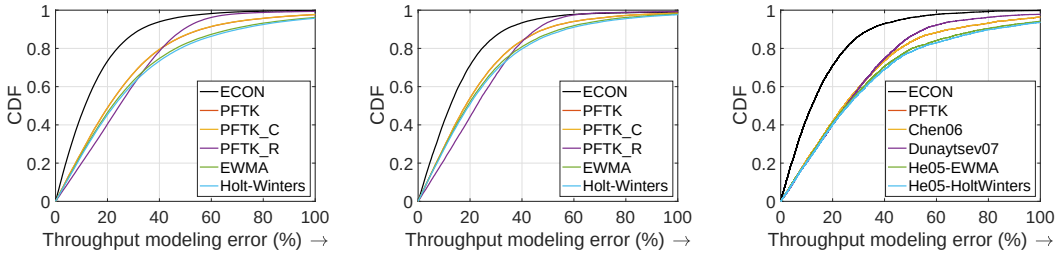
We compare our model accuracy with other models published in the literature.

- The classic **PFTK** model [57] (discussed in §2.2).



(a) Throughput (Azure) (b) Throughput (Southeast) (c) Throughput (Northeast)

Figure 5. CDF of TCP CUBIC throughput modeling error under all eligible models for the Azure, Southeast, and Northeast networks. Some alternate analytical models [22, 24, 57] are designed for TCP Reno, so we are unable to compare against them for the CUBIC experiments. The experiments predict throughput for the next 1 second.



(a) Throughput (Azure) (b) Throughput (Southeast) (c) Throughput (Northeast)

Figure 6. CDF of TCP Reno throughput modeling error under all models for the Azure, Southeast, and Northeast networks. The experiments predict throughput for the next 30 seconds.

- The model proposed by Chen et al. [22], denoted as **Chen06**, that corrects errors in the classic PFTK model relating to overprediction of the send rate.
- The model proposed by Dunaytsev et al. [24], denoted as **Dunaytsev07**, that predicts TCP Reno's throughput by accounting for fast retransmit/fast recovery dynamics and slow start state.
- Exponentially weighted moving average (**He05-EWMA**) [36], a parameterized history-based model that predicts throughput based on previously observed throughput values.
- Holt-Winters (**He05-HoltWinters**), also a parameterized history-based model that is well suited to non-stationary processes. He05-HoltWinters predicts throughput by capturing the trend in previously observed throughput values, as detailed in He at al. [36].

The PFTK model [57] was published before 2004, when a draft of the CUBIC variant was first introduced [59]; as such, the first three (PFTK-based) models only apply to TCP Reno, and not CUBIC. We thus compare our CUBIC model with He05-EWMA and He05-HoltWinters. For our TCP Reno model, we compare with all five alternative models.

Sensitivity analysis

For a fair comparison, we enhance the three analytical models [22, 24, 57] to use the same sliding window approach as our model (§4.3). He05-EWMA and He05-HoltWinters already use a sliding window approach.

For each model, we run a sensitivity analysis across different sliding window sizes, or training size (number of seconds of historical data to use), and the update frequency (how often to slide the window). We vary both parameters between 10s and 100s; values above 100s result in stale data, and values closer to 1s result in high overhead. For the formula-based models (PFTK, Chen06, Dunaytsev07, and ECON), we find that a large window size works well, so we set the window size to 100s. For He05-EWMA and He05-HoltWinters, we find that a smaller window size works well, so we set 10s for these models. In all cases, we find that the update frequency of 10s works well.

The He05-EWMA and He05-HoltWinters also have two additional parameters α and β that need tuning. The original paper used the values of $\alpha = 0.8$ and $\beta = 0.2$ [36]. However, our sensitivity analysis showed that $\alpha = 0.8$ and $\beta = 0.8$ works best on our testbed, so we use these parameter values. We verify that using different parameter values result in poorer prediction performance.

5.2 TCP CUBIC and Reno throughput results

We evaluate the prediction error when running two TCP variants: TCP CUBIC and Reno. We predict the throughput for the next 1s, 5s, and 30s. Below we present the prediction errors when using TCP CUBIC and predicting for the next 1s

and TCP Reno when predicting for the next 30s. The other combinations showed quantitatively similar results.

CDF of TCP Cubic Throughput modeling errors
Figure 5 shows the CDF of TCP CUBIC throughput modeling errors for all models and all networks. These results are based on 75,000 prediction windows across all networks. We compute the modeling error by comparing our predictions with the observed experimental values.

Our model significantly outperforms all other models for all networks. Our median throughput modeling error for Azure, Southeast, and Northeast is about 14.5%, 17.5%, and 13.5%, respectively. By comparison, the best median throughout modeling error for He05-EWMA/He05-HoltWinters is 27.3%, 51.1%, and 33.2%, for the three networks.

For the alternative models, the error is large for the *Southeast* network. This is because this network has a small RTT, resulting in a very dynamic cwnd evolution, which is hard to capture using historical observations alone. We find that He05-EWMA typically has (slightly) lower errors compared to He05-HoltWinters.

When comparing the *average modeling error*, ECON outperforms the best alternative model in each case by about 65% or 29% absolute reduction in error (averages not shown in figure).

CDF of TCP Reno Throughput modeling errors
We evaluate the modeling error for TCP Reno over all three networks and compare with all five alternative models as they apply to TCP Reno. The modeling error corresponds to predicting the throughput for a 30 second interval.

Figure 6 shows the CDF of TCP Reno throughput modeling error for all networks. As before, ECON outperforms all given alternatives.

Modeling errors: Varying parameters, dynamic network conditions
Figure 7 breaks down the TCP CUBIC throughput modeling error as a function of the starting congestion window (scwnd) for our model and the alternative models under the *Azure* network. While we outperform other models for all scwnd values, the relative improvement is more pronounced for smaller and larger scwnd values. This is because other models assume that the loss probability (p) is independent of cwnd, whereas we find that p is higher than average when cwnd is either small or large (see Figure 2).

We evaluate ECON by varying the number of parallel connections (npc). Our model outperforms other models irrespective of the number of parallel connections; further, our 75%ile error numbers are much lower than the corresponding numbers for other models (graphs omitted for brevity).

To evaluate the effectiveness of the sliding window approach across models, we consider the Azure network with the server VM located in West US 2 and run TCP Reno. We use TC [1] to change the RTT and loss rate, p, to emulate varying network conditions. In each experiment, we start

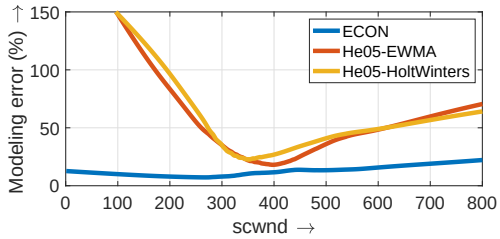


Figure 7. Throughput modeling error versus the starting congestion window size (scwnd) for Azure network.

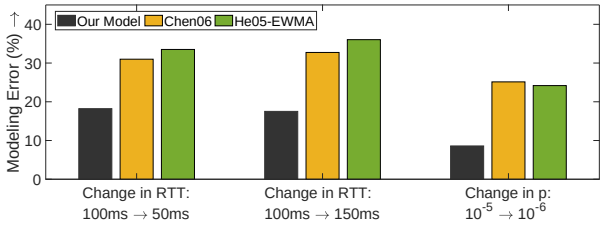


Figure 8. Throughput modeling error under changing network conditions. Our model continues to outperform other models.

with a 100ms RTT and $p = 10^{-5}$. Then, a few minutes into the experiment, we abruptly change the network conditions; we terminate the experiment after 30 minutes.

Figure 8 shows the average modeling error for the case of decreasing RTT, increasing RTT, and decreasing p. We compare with the two best performing alternatives. Our model results in much lower modeling error compared to other models in all cases.

5.3 Modeling results for wireless network
For wireless experiments, we use the wired VM in the *Southeast* CloudLab site as the sender and then set the receiver in the *Northeast* to be on a wireless (WiFi) connection. We set the number of TCP connections to be 1, and run experiments for a total of 5 hours.

Figure 9 shows the CDF of the throughout modeling error under TCP Reno (so we can compare with all alternative models); results are similar under CUBIC. Compared to the best alternative model, our model reduces the median and average throughput modeling error from 14.9% and 16.8% (He05-HoltWinters) to 3.3% and 6.9%, respectively.

For the wireless network, the p(cwnd) plot (not shown) continues to exhibit a U-shaped relationship. However, due to the larger RTT for wireless connections, we observe a narrower range of congestion window values, resulting in lower errors across all models compared to the wired networks.

5.4 Latency modeling errors
An immediate and practical application of our TCP throughput model is to predict the latency of TCP flows; TCP is often used as the underlying transport layer for several applications, so predicting the TCP latency of data transfers is beneficial for application-level modeling.

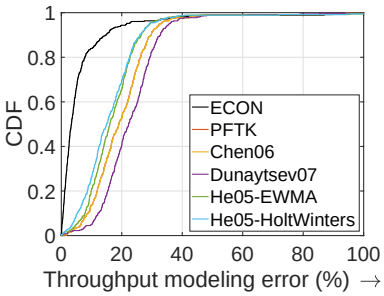
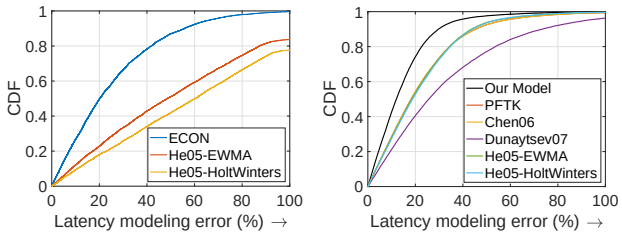


Figure 9. TCP Reno throughput modeling error for a wireless connection on Southeast network.



(a) TCP Cubic, Southeast (b) TCP Reno, Southeast

Figure 10. TCP latency modeling error for (a) CUBIC and (b) Reno, on Southeast network.

Network	ECON	EWMA	H.Winters
Azure	16.8	25.0	33.0
SouthEast	20.2	48.5	60.2
NorthEast	15.0	32.7	40.1

Table 2. Average error for CUBIC latency modeling for all networks.

Figure 10 shows the latency prediction error CDF under TCP CUBIC and TCP Reno for the *Southeast* network. The results are based on more than 5000 data transfer experiments, with the data size in each transfer ranging from 5MB to 250MB. While some of the alternative models, such as He05-EWMA [36], only focus on throughput modeling, we extend their models to also predict latency, similar to our model. Data transfers of size less than 5MB often completed within one RTT and were thus not considered as predicting this latency is straightforward for all models.

The median latency prediction error of our model under CUBIC is about 12%, whereas that of other models is about 58%. Under Reno, our median error is about 11%, while that of other models ranges from about 28% (for He05-HoltWinters) to 35% (for Dunaytsev07). We also find that the worst case error for our model is about 70%, whereas that for other models is greater than 100%.

Figure 11 shows the average prediction error under TCP Reno broken down by data size to be transferred, along with the 25%ile and 75%ile bars; we focus on Reno so we can compare with all alternative models. The prediction error

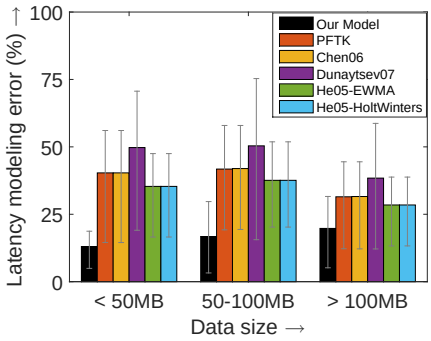


Figure 11. Latency prediction error for different data size ranges. Our model provides greater benefits for lower data sizes.

improvement afforded by our model over other models decreases as the data size increases. This is to be expected as (i) we use the predicted throughput of the first 1s interval as a proxy for the entire transfer lifetime, and (ii) other analytical models, such as PFTK, use steady-state analysis with constant loss rates which is well suited for bulk transfers and not short flows [57].

Finally, Table 2 shows the average latency prediction error under TCP CUBIC for all networks and all eligible models. Our model consistently outperforms all other models for *all* networks. Results are similar under TCP Reno, with the average modeling error across all networks under ECON being around 16%, whereas that under the best alternative model (He05-EWMA) being around 30%.

6 Web: HTTP/1.1 vs HTTP/2

HTTP/2 was designed to improve performance over the HTTP/1.1 protocol. However, it is not always clear when HTTP/2 outperforms HTTP/1.1, as shown by empirical studies [60, 73]. Our goal is to analytically evaluate the performance of HTTP/1.1 and HTTP/2 for a given workload and environment condition, to decide which application protocol to use in a given scenario. We use ECON's HTTP-level models from §4.2 to accurately predict the throughput and latency of HTTP/1.1 and HTTP/2.

6.1 Experimental setup for validation

To validate our HTTP models, we experiment with HTTP traffic between a browser and a Web server in Northeast US. We use the Chrome browser (v63.0.3239.132) and an Nginx Web server (v1.10.3) on Ubuntu 16.04.3 LTS OS. The browser and the Web server support HTTP/1.1 and HTTP/2. For HTTP/1.1, we use the default value of 6 parallel and persistent connections. For HTTP/2, Nginx sets the default value of 128 maximum concurrent streams. All objects are requested from a single server. We use TC [1] to vary the network conditions. We configure the Chrome browser to automatically request a certain number and size of objects, as specified by the experiment.

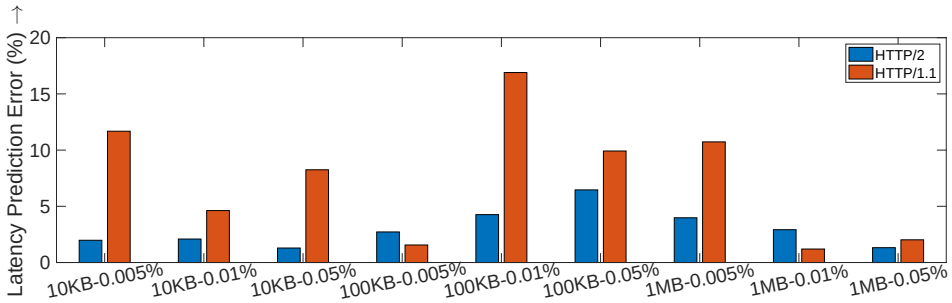


Figure 12. HTTP/1.1 and HTTP/2 latency modeling error under different file sizes and loss rates. For the first three bars the transfers are completed within the slow-start phase of TCP. The low errors show that the ECON's prediction in slow-start phase is accurate.

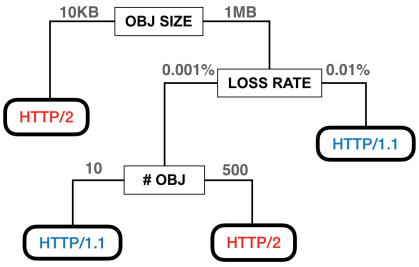


Figure 13. Prediction results for HTTP/1.1 vs. HTTP/2.

6.2 HTTP prediction results

Figure 12 shows our latency modeling error for HTTP/1.1 and HTTP/2 across 9 different experiments. In each experiment, the browser requests 500 objects; we vary the object sizes (10KB, 100KB, 1MB) and loss probability (0.005%, 0.01%, 0.05%) across experiments. Our average modeling error for HTTP/1.1 and HTTP/2 is 3.00% and 7.43%, respectively. When fetching 10KB objects, both HTTP/1.1 and HTTP/2 take less than the TCP remains in the slow-start phase, showing that ECON can accurately predict performance even when TCP is in slow start phase.

6.3 HTTP/1.1 versus HTTP/2

We now return to the key question we posed earlier—should we use HTTP/1.1 or HTTP/2 for a given environment and workload. Prior work [73] has investigated this question via large-scale empirical studies. We now show that our HTTP models can analyze different scenarios and provide similar conclusions as the prior work, but without requiring extensive experimentation.

We explore 16 different scenarios of workload and network conditions: object size (10KB, 1MB), loss probability (10^{-5}, 10^{-4}), number of objects (10, 500), and RTT (50ms, 200ms) For each scenario, we predict the latency under HTTP/1.1 and HTTP/2 using our models. Figure 13 illustrates our prediction results as a tree diagram that determines when HTTP/1.1 outperforms HTTP/2 and vice-versa. For all 16 scenarios, our results are in agreement with the empirical results obtained by prior work [73], without requiring extensive experimental data.

In most cases, HTTP/2 has lower predicted latency compared to HTTP/1.1. However, under high loss probability, high RTTs, and large file sizes, HTTP/1.1 has lower latency, as shown in Figure 13. We find that the wrong choice between HTTP/2 and HTTP/1.1 can result in up to 4.6× higher latency, highlighting the importance of accurate analysis. We find that the average increase in latency, across all 16 scenarios, when making the wrong choice is about 264%.

7 Applications to Data Centers

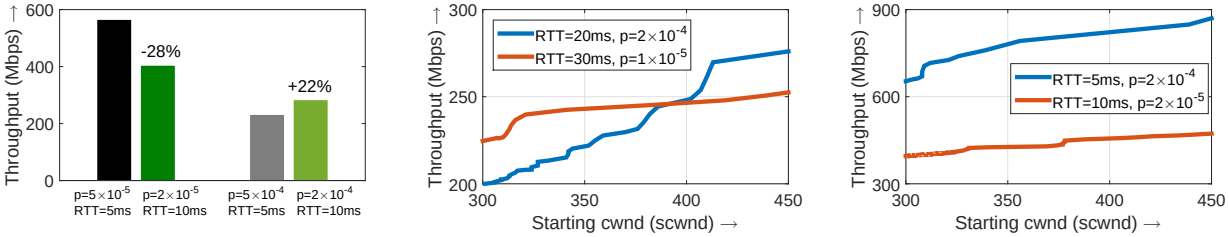
TCP traffic continues to dominate in data centers [8]. For example, data transfer between web or application servers and MySQL databases is over TCP (Rest APIs use HTTP, which in turn relies on TCP), as is the data replication traffic in databases (such as Cassandra [65]) [30, 33, 69] and the cloud management and monitoring traffic (such as OpenStack's Ceilometer [56]) in cloud data centers. Our TCP models can thus be readily applied to data center applications for performance management.

One concrete use case we study is that of determining the best source/destination server for data transfer. Several data processing and data storage frameworks replicate data on multiple nodes for availability and reliability [11, 23, 67]. In such cases, when an external application wants to read some data from these sources, it has a choice to make – "which node to read data from?". Likewise, in several scenarios, there is a choice of "which node to send data to", as is the case for forwarding a request to one of many web/application servers and when transferring data collected from a server to other nodes for analysis [31, 32, 66]. We refer to these questions as the data transfer path selection problem.

The main challenge here is that different paths may have different network characteristics, such as RTT, loss rate, etc. While one can empirically profile the network performance on each path and pick the one with the best historical performance, this approach ignores the dynamic and unpredictable congestion behavior in TCP. By contrast, our TCP models are designed to address such dynamic behaviors and can quickly determine the best path for a given data transfer.

7.1 Experimental setup

We use the Azure public cloud data center, as detailed in §5.1, for our experiments. We use 8 VMs, all in the East US region of Azure. We use *iPerf* [68] to emulate data transfer,



(a) RTT and loss rate alone are not enough to choose the best path.

(b) scwnd value matters when determining the best path.

(c) For specific RTT and loss rate values, scwnd's effect may be small.

Figure 14. Impact of various parameters on the choice of optimal path. y axes in (b) and (c) do not start at 0 for clearer illustration.

and vary network conditions using TC [1] to create different data transfer paths.

7.2 Choosing the optimal data transfer path

To choose the best source/destination node for data transfer, we leverage our models to predict throughput based on the network conditions (RTT, loss probability, etc.) of each path. Recall, from §5.2, that the average prediction error for our Azure setup is about 16%.

Figure 14(a) shows our predicted throughput for four specific data transfer paths with different RTTs and loss probabilities (p). In the first set of bars (left), we see that throughput drops by 28% when the RTT doubles (from 5ms to 10ms) despite a 2.5× drop in p (from 5×10^{-5} to 2×10^{-5}). While this observation is interesting in itself, we find that this is not always the case. The second set of bars (right) show a similar pair of paths with a 2× rise in RTT but a 2.5× drop in p, except this time p drops from 5×10^{-4} to 2×10^{-4}. Interestingly, the higher RTT path has 22% *higher* throughput compared to the lower RTT path, contrary to our previous observation. Using a purely empirical approach to predict the relative ordering of throughput for the second case based on observations from the first case would result in a significant loss of performance (about 52Mbps lower throughput).

To further analyze the behavior of throughput, we consider the starting congestion window (scwnd) value of TCP flows. Figure 14(b) shows the predicted throughput as a function of scwnd for two paths, one with RTT=20ms and $p = 2\times10^{-4}$, and the other with RTT=30ms (a 1.5× increase) and $p = 1\times10^{-5}$ (a 20× decrease); the x-axis and y-axis limits are intentionally chosen to clearly illustrate the comparison of throughputs. Interestingly, we see that the relative ordering of throughput *depends* on scwnd. While the lower RTT path has higher throughput for most of the scwnd range, the order *reverses* for the lower scwnd range.

Figure 14(c) shows our results for yet another scenario where we compare a 5ms RTT, $p = 2\times10^{-4}$ path with a 10ms RTT, $p = 2\times10^{-5}$ path. In this case, the lower RTT path clearly dominates, by as much 2×, within the chosen range of scwnd. In fact, the throughput for the lower RTT path at scwnd= 300 is still higher than the throughput for the higher RTT path at scwnd= 450.

Relying only on empirical observations to choose the best path can be insufficient, as demonstrated by our non-intuitive results above. By using our prediction models, practitioners can quickly decide which data transfer path to use.

8 Related Work

Existing work on TCP modeling can be broadly categorized [36] into (i) formula-based (FB) models and (ii) history-based (HB) models.

FB predictors rely on analytical models to characterize the TCP performance as a function of the underlying network [19, 22, 24, 53, 57, 58]. HB predictors, on the other hand, employ time-series based techniques to forecast TCP performance based on historically observed values [36, 63, 70]. Our model leverages the advantages of both FB and HB models by augmenting analytical modeling with dynamically updated empirical information.

FB models: The PFTK model [57] characterizes the steady-state TCP (Reno) throughput as a function of loss probability (p) and RTT. Newer models correct errors in PFT [22] and accounting for fast retransmit/fast recovery dynamics and slow start phase [24]. Cardwell et al. extend the PFTK throughput model to TCP latency modeling [19].

Several other stochastic TCP modeling approaches have been proposed but assume specific distributions for packet losses [7, 9, 53].

Goyal et al. predict the TCP throughput under a random loss model (constant loss probability) based on monitored metrics sampled at the congestions points on the path [34], but assume that the congestion points in the link are known. Hespanha et al. [38] propose a theoretical model for TCP performance when operating under the drop-tail queueing policy. Fortin-Parisi and Sericola use Markov chains to model TCP performance [28]; however, like PFTK, they assume a fixed packet loss probability.

HB models: Exponential weighted moving average (He05-EWMA) and Holt-Winters (He05-HoltWinters) [36] are HB models that predict TCP throughput based on previously observed throughput values (see §5.1).

Prior approaches have also employed learning algorithms, such as Support Vector Regression, for TCP performance modeling [15, 44, 52]. However, such approaches typically

require additional features that are not always observable, such as queueing delay and available bandwidth [52], and may also require non-trivial parameter tuning [15].

Models for specific scenarios and variants: There are several works that focus specifically on short flows [18, 50] but also assume that losses are independent. For large flows, DualPats leverages the correlation between TCP throughput and flow size to predict performance [47]. There are also works that focus specifically on deriving models for parallel TCP connections [10, 48] and for wireless TCP connections [29, 41, 42, 55]. By contrast, we capture the TCP performance under all the above scenarios.

Some models use router-level information to model TCP performance under different AQM techniques such as RED [16, 45, 54]. However, router-level information is not easy to obtain. Instead, in our work, `p(cwnd)` is derived by observing the end-to-end performance, which incorporates the effect of different queue management techniques.

Given that TCP has many variants, several studies [13, 25, 46, 58, 61, 62, 72] focus on modeling TCP throughput under different congestion control algorithms, including Reno, NewReno, Vegas, CUBIC, Tahoe, and Fast. We show in §3 that our model can characterize TCP performance under both Reno and CUBIC. We believe that our model can be extended to other TCP variants, such as BBR [17], that have a specifiable cwnd evolution; we will investigate such model extensions as part of future work.

HTTP models: Zarifis et al. recently proposed an approach to estimate HTTP/2 performance [75]. However, this approach relies on the availability of existing HTTP/1.1 traces. Heidemann et al. [37] propose an analytical model for HTTP performance over different networks and transport protocols; however, they assume no packet loss. Finally, there are works that focus on HTTP traffic modeling [20, 49], and not throughput modeling, which is the focus of our work.

9 Conclusion

ECON presents a scalable and systematic model to easily evaluate the performance of different network choices in the context of HTTP and data center applications. The core of ECON is an analytical TCP model that adapts to dynamic network conditions by empirically estimating the effect of congestion window on loss rate. By taking into account the effect of network congestion, ECON removes the limiting assumptions made by existing analytical models. ECON's application level model for latency and HTTP is built on top of the TCP model.

Evaluation results across three different wired networks and one wireless network highlight the accuracy of the ECON TCP model. The ECON latency predictions and HTTP predictions have an error of less than 15%. We demonstrate ECON's applicability to improving network performance by accurately and scalably predicting (i) the network and workload conditions under which HTTP/2 outperforms HTTP/1.1 and vice-versa, and (ii) the optimal path for data transfer in public clouds, among paths with different network characteristics.

A Appendix: TCP Modeling Details

Deriving $E[M]$: To determine M, the index of the first lost packet, we first consider a single connection. The probability of a loss in the very first packet is simply $p(scwnd)$; thus, $Pr(M = 1) = p(scwnd)$. The probability of having the first loss at the second packet, assuming $scwnd > 1$, is $Pr(M = 2) = (1 - p(scwnd)) \cdot p(scwnd)$, since the first packet did not experience a loss; similarly for $M \leq scwnd$.

For $M > scwnd$, we consider the congestion avoidance algorithm. Let $cwnd_j$ be the $cwnd$ for the j^{th} congestion window, with $cwnd_1 = scwnd$. Under Reno, $cwnd_j = scwnd+j-1$, and under CUBIC, $cwnd_j = C((j-1)RTT-K)^3+scwnd/(1-\beta)$. As $cwnd$ increases, the loss probability function changes based on $p(cwnd)$. Thus, for $scwnd < i \leq cwnd_2$:

$$Pr(M = i) = (1 - p(scwnd))^{scwnd} \cdot (1 - p(cwnd_2))^{i-1} \cdot p(cwnd_2)$$

We similarly compute $Pr(M = i)$ based on $p(cwnd)$ for all i and derive $E[M] = \sum_i i \cdot Pr(M = i)$.

We repeat the same analysis as above by replacing $p(cwnd)$ with $p(cwnd, npc)$. Substituting for $Pr(M = i)$ in $E[M] = \sum_i i \cdot Pr(M = i)$ and simplifying results in Eq. (3).

Deriving $E[X]$: Since each RTT corresponds to one $cwnd$, we have $E[X] = \sum_{n=1}^{\infty} n \cdot q(n, npc)$, where $q(n, npc)$ is the probability of the first packet loss occurring during the n^{th} $cwnd$ after the transfer starts. To derive $q(n, npc)$, we first derive the probability that there is no packet loss in a $cwnd$ of size s given npc connections as $\overline{q}_{s,npc} = (1 - p(s, npc))^s$. Thus, the probability of a loss in a cwnd of size s is $q_{s,npc} = 1 - \overline{q}_{s,npc}$. Now, $q(n, npc)$ can be expressed in terms of $q_{s,npc}$ as $q(n, npc) = q_{cwnd_n,npc} \cdot \prod_{j=1}^{n-1} \overline{q}_{cwnd_j,npc}$. Finally, $E[X]$ can be obtained as:

$$\begin{aligned}\sum_{n=1}^{\infty} n \cdot q(n, npc) \quad &= \sum_{n=1}^{\infty} n(1 - (1 - p(cwnd_n, npc))^{cwnd_n}) \\ &\quad \times \prod_{j=1}^{n-1}(1 - p(cwnd_j, npc))^{cwnd_j}\end{aligned}$$

Deriving $E[N]$: If we assume that the first loss occurs at the m_{th} packet of the n_{th} $cwnd$, then $M = \sum_{i=0}^{n-2}(scwnd + i) + m$. N includes all M packets except the lost packet (M-1), and the additional packets sent before the loss is detected by the sender ($cwnd_{E[X]}$). Thus, $E[N] = (E[M] - 1) + cwnd_{E[X]}$.

Slow-start phase: Consider the first packet loss to occur at the m_{th} packet of the i_{th} RTT. The number of successfully transferred packets is $icwnd \times (2^{i-1} - 1) + m - 1$. Let M_s denote the index of the first packet loss during slow start, then:

$$\begin{aligned}E[M_s] \quad = \quad &\sum_{i=1}^{\infty} \sum_{m=1}^{icwnd \cdot 2^{i-1}} (icwnd \times (2^{i-1} - 1) + m) \\ &\times \prod_{j=1}^{i-1}(1 - p(icwnd \times 2^{j-1}, npc))^{icwnd \times 2^{j-1}} \\ &\times (1 - p(icwnd \times 2^{i-1}, npc))^{m-1} \cdot p(icwnd \times 2^{i-1}, npc)\end{aligned}$$

Total RTTs during the slow start is $E[X_s] = \lfloor \log_2(E[M_s])/icwnd \rfloor + 1$. The total number of packets sent during slow start is thus $E[N_s] = E[M_s] + icwnd \times 2^{E[X_s]-1} - 1$.

References

[1] Linux Traffic Controller. http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html.

[2] TCP NewReno. https://tools.ietf.org/html/rfc6582.

[3] TCP Reno. https://intronetworks.cs.luc.edu/current/html/reno.html.

[4] Tcp probe. https://wiki.linuxfoundation.org/networking/tcpprobe, 2016.

[5] Apache. https://httpd.apache.org/, 2017.

[6] nginix. https://nginx.org/en/, 2017.

[7] Abouzeid, A. A., Roy, S., and Azizoglu, M. Stochastic modeling of tcp over lossy links. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2000), vol. 3, IEEE, pp. 1724–1733.

[8] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference* (New York, NY, USA, 2010), SIGCOMM '10, ACM, pp. 63–74.

[9] Altman, E., Avrachenkov, K., and Barakat, C. A stochastic model of tcp/ip with stationary random losses. *ACM SIGCOMM Computer Communication Review 30*, 4 (2000), 231–242.

[10] Altman, E., Barman, D., Tuffin, B., and Vojnovic, M. Parallel tcp sockets: Simple model, throughput and validation. In *INFOCOM* (2006), vol. 2006, pp. 1–12.

[11] Amur, H., Cipar, J., Gupta, V., Ganger, G. R., Kozuch, M. A., and Schwan, K. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (Indianapolis, IN, USA, 2010), SoCC '10, pp. 217–228.

[12] Balakrishnan, H., Padmanabhan, V. N., Seshan, S., Stemm, M., and Katz, R. H. Tcp behavior of a busy internet server: Analysis and improvements. In *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (1998), vol. 1, IEEE, pp. 252–262.

[13] Bao, W., Wong, V. W., and Leung, V. C. A model for steady state throughput of tcp cubic. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE* (2010), IEEE, pp. 1–6.

[14] Belshe, M., Thomson, M., and Peon, R. Hypertext transfer protocol version 2 (http/2).

[15] Bermolen, P., and Rossi, D. Support vector regression for link load prediction. *Computer Networks 53*, 2 (2009), 191–201.

[16] Bu, T., and Towsley, D. Fixed point approximations for tcp behavior in an aqm network. *SIGMETRICS Perform. Eval. Rev. 29*, 1 (June 2001), 216–225.

[17] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and Jacobson, V. Bbr: Congestion-based congestion control. *Queue 14*, 5 (Oct. 2016), 50:20–50:53.

[18] Cardwell, N., Savage, S., and Anderson, T. Modeling the performance of short tcp connections. *Techical Report* (1998).

[19] Cardwell, N., Savage, S., and Anderson, T. Modeling tcp latency. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2000), vol. 3, IEEE, pp. 1742–1751.

[20] Casilari, E., Gonzblez, F., and Sandoval, F. Modeling of http traffic. *IEEE Communications Letters 5*, 6 (2001), 272–274.

[21] Chan, M. C., and Ramjee, R. Tcp/ip performance over 3g wireless links with rate and delay variation. *Wireless Networks 11*, 1-2 (2005), 81–97.

[22] Chen, Z., Bu, T., Ammar, M., and Towsley, D. Comments on modeling tcp reno performance: a simple model and its empirical validation. *IEEE/ACM Transactions on Networking (ToN) 14*, 2 (2006), 451–453.

[23] Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6* (Berkeley, CA, USA, 2004), OSDI'04, USENIX Association, pp. 137–150.

[24] Dunaytsev, R., Koucheryavy, Y., and Harju, J. The pftk-model revised. *Computer communications 29*, 13 (2006), 2671–2679.

[25] Dunaytsev, R., Koucheryavy, Y., and Harju, J. Tcp newreno throughput in the presence of correlated losses: The slow-but-steady variant. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings* (2006), IEEE, pp. 1–6.

[26] Fall, K. R., and Stevens, W. R. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.

[27] Fitzpatrick, B. Distributed caching with memcached. *Linux Journal 2004*, 124 (2004).

[28] Fortin-Parisi, S., and Sericola, B. A markov model of tcp throughput, goodput and slow start. *Performance Evaluation 58*, 2-3 (2004), 89–108.

[29] Fu, S., and Atiquzzaman, M. Modelling tcp reno with spurious timeouts in wireless mobile environments. In *Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on* (2003), IEEE, pp. 391–396.

[30] Gandhi, A., Chen, Y., Gmach, D., Arlitt, M., and Marwah, M. Minimizing Data Center SLA Violations and Power Consumption via Hybrid Resource Provisioning. In *Proceedings of the 2011 International Green Computing Conference* (Orlando, FL, USA, 2011), IGCC '11, pp. 49–56.

[31] Gandhi, A., Harchol-Balter, M., Raghunathan, R., and Kozuch, M. AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *Transactions on Computer Systems 30* (2012).

[32] Gandhi, A., Zhu, T., Harchol-Balter, M., and Kozuch, M. SOFTScale: Scaling Opportunistically For Transient Scaling. In *Proceedings of the 13th International Middleware Conference* (Montreal, Quebec, Canada, 2012), Middleware '12, pp. 142–163.

[33] Gmach, D., Rolia, J., Cherkasova, L., and Kemper, A. Resource pool management: Reactive versus proactive or let's be friends. *Computer Networks 53*, 17 (Dec. 2009), 2905–2922.

[34] Goyal, M., Guerin, R., and Rajan, R. Predicting tcp throughput from non-invasive network sampling. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2002), vol. 1, IEEE, pp. 180–189.

[35] Ha, S., Rhee, I., and Xu, L. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev. 42*, 5 (July 2008), 64–74.

[36] He, Q., Dovrolis, C., and Ammar, M. On the predictability of large transfer tcp throughput. In *ACM SIGCOMM Computer Communication Review* (2005), vol. 35, ACM, pp. 145–156.

[37] Heidemann, J., Obraczka, K., and Touch, J. Modeling the performance of http over several transport protocols. *IEEE/ACM Transactions on Networking (TON) 5*, 5 (1997), 616–630.

[38] Hespanha, J. P., Bohacek, S., Obraczka, K., and Lee, J. Hybrid modeling of tcp congestion control. In *International Workshop on Hybrid Systems: Computation and Control* (2001), Springer, pp. 291–304.

[39] HTTP/2. https://http2.github.io/.

[40] Jiang, H., and Dovrolis, C. Passive estimation of tcp round-trip times. *ACM SIGCOMM Computer Communication Review 32*, 3 (2002), 75–88.

[41] Katsuhiro, N., Okada, H., Yamazato, T., Katayama, M., and Ogawa, A. New analytical model for the tcp throughput in wireless environment. In *Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd* (2001), vol. 3, IEEE, pp. 2128–2132.

[42] Kim, M., Médard, M., and Barros, J. Modeling network coded tcp throughput: A simple model and its validation. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools* (2011), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 131–140.

[43] Lai, K., and Baker, M. Measuring bandwidth. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (1999), vol. 1, IEEE, pp. 235–245.

[44] Lee, C., Abe, H., Hirotsu, T., and Umemura, K. Analytical modeling of network throughput prediction on the internet. *IEICE TRANSACTIONS on Information and Systems 95*, 12 (2012), 2870–2878.

[45] Low, S. H. A duality model of tcp and queue management algorithms. *IEEE/ACM Trans. Netw. 11*, 4 (Aug. 2003), 525–536.

[46] Low, S. H., Peterson, L. L., and Wang, L. Understanding tcp vegas: a duality model. *Journal of the ACM (JACM) 49*, 2 (2002), 207–235.

[47] Lu, D., Qiao, Y., Dinda, P. A., and Bustamante, F. E. Characterizing and predicting tcp throughput on the wide area network. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on* (2005), IEEE, pp. 414–424.

[48] Lu, D., Qiao, Y., Dinda, P. A., and Bustamante, F. E. Modeling and taming parallel tcp on the wide area network. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International* (2005), IEEE, pp. 10–pp.

[49] Mah, B. A. An empirical model of http network traffic. In *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE* (1997), vol. 2, IEEE, pp. 592–600.

[50] Mellia, M., and Zhang, H. Tcp model for short lived flows. *IEEE communications letters 6*, 2 (2002), 85–87.

[51] Microsoft Azure. Azure regions. https://azure.microsoft.com/en-us/regions.

[52] Mirza, M., Sommers, J., Barford, P., and Zhu, X. A machine learning approach to tcp throughput prediction. In *ACM SIGMETRICS Performance Evaluation Review* (2007), vol. 35, ACM, pp. 97–108.

[53] Misra, V., Gong, W.-B., and Towsley, D. Stochastic differential equation modeling and analysis of tcp-windowsize behavior. In *Proceedings of PERFORMANCE* (1999), vol. 99.

[54] Misra, V., Gong, W.-B., and Towsley, D. Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (New York, NY, USA, 2000), SIGCOMM '00, ACM, pp. 151–160.

[55] Nguyen, G. T., Katz, R. H., Noble, B., and Satyanarayanan, M. A trace-based approach for modeling wireless channel behavior. In *Proceedings of the 28th conference on Winter simulation* (1996), IEEE Computer Society, pp. 597–604.

[56] Openstack.org. Ceilometer. https://docs.openstack.org/ceilometer.

[57] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J. Modeling tcp throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review 28*, 4 (1998), 303–314.

[58] Parvez, N., Mahanti, A., and Williamson, C. An analytic throughput model for tcp newreno. *IEEE/ACM Transactions on Networking (ToN) 18*, 2 (2010), 448–461.

[59] Rhee, I., and Xu, L. CUBIC: A New TCP-Friendly High-Speed TCP Variant. In *Proceedings of the 3rd International Workshop on Protocols for Fast Long-Distance Networks* (Lyon, France, 2005), PFLDnet Workshop.

[60] Rosen, S., Han, B., Hao, S., Mao, Z. M., and Qian, F. Push or request: An investigation of http/2 server push for improving mobile performance. In *Proceedings of the 26th International Conference on World Wide Web* (Republic and Canton of Geneva, Switzerland, 2017), WWW '17, International World Wide Web Conferences Steering Committee, pp. 459–468.

[61] Samios, C. B., and Vernon, M. K. Modeling the throughput of tcp vegas. In *ACM SIGMETRICS Performance Evaluation Review* (2003), vol. 31, ACM, pp. 71–81.

[62] Sikdar, B., Kalyanaraman, S., and Vastola, K. S. Analytic models for the latency and steady-state throughput of tcp tahoe, reno, and sack. *IEEE/ACM Transactions On Networking 11*, 6 (2003), 959–971.

[63] Swany, M., and Wolski, R. Multivariate resource performance forecasting in the network weather service. In *Supercomputing, ACM/IEEE 2002 Conference* (2002), IEEE, pp. 11–11.

[64] The Apache Software Foundation. Apache Hadoop. http://hadoop.apache.org.

[65] The Apache Software Foundation. Cassandra. http://cassandra.apache.org/doc/latest/configuration/cassandra_config_file.html.

[66] The Apache Software Foundation. Kafka. https://kafka.apache.org.

[67] Thereska, E., Donnelly, A., and Narayanan, D. Sierra: practical power-proportionality for data center storage. In *Proceedings of the 6th European Conference on Computer Systems* (Salzburg, Austria, 2011), EuroSys '11, pp. 169–182.

[68] Tirumala, A., Qin, F., Dugan, J., Ferguson, J., and Gibbs, K. Iperf.

[69] Urgaonkar, B., and Chandra, A. Dynamic Provisioning of Multi-tier Internet Applications. In *Proceedings of the 2nd International Conference on Automatic Computing* (Seattle, WA, USA, 2005), ICAC '05, pp. 217–228.

[70] Vazhkudai, S., Schopf, J. M., and Foster, I. Predicting the performance of wide area data transfers. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM* (2001), IEEE, pp. 10–pp.

[71] Veal, B., Li, K., and Lowenthal, D. New methods for passive estimation of tcp round-trip times. In *International Workshop on Passive and Active Network Measurement* (2005), Springer, pp. 121–134.

[72] Wang, J., Wei, D. X., Choi, J.-Y., and Low, S. H. Modelling and stability of fast tcp. In *Wireless Communications*. Springer, 2007, pp. 331–356.

[73] Wang, X. S., Balasubramanian, A., Krishnamurthy, A., and Wetherall, D. How speedy is spdy? In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2014), NSDI'14, USENIX Association, pp. 387–399.

[74] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing* (Boston, MA, USA, 2010), HotCloud '10.

[75] Zarifis, K., Holland, M., Jain, M., Katz-Bassett, E., and Govindan, R. Modeling http/2 speed from http/1 traces. In *International Conference on Passive and Active Network Measurement* (2016), Springer, pp. 233–247.