ECON: Modeling the network to improve application performance

Yi Cao
Stony Brook University
yicao1@cs.stonybrook.edu

Javad Nejati
Stony Brook University
jnejati@cs.stonybrook.edu

Aruna Balasubramanian
Stony Brook University
arunab@cs.stonybrook.edu

Anshul Gandhi
Stony Brook University
anshul@cs.stonybrook.edu

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, and uses the TCP model as a building block for predicting application performance. ECON infers a relationship between loss and congestion using empirical data that drives an online model to predict TCP performance. ECON then builds on the TCP model to predict latency and HTTP performance. Across four wired and one wireless network, our model outperforms seven alternative TCP models. We demonstrate how ECON (i) can be used by a Web server application to choose between HTTP/1.1 and HTTP/2 for a given Web page and network condition, and (ii) can be used by a video application to choose the optimal bitrate that maximizes video quality without rebuffering.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Application layer protocols**; **Network performance modeling**.

ACM Reference Format:
Yi Cao, Javad Nejati, Aruna Balasubramanian, and Anshul Gandhi. 2019. ECON: Modeling the network to improve application performance. In *Internet Measurement Conference (IMC '19), October 21–23, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3355369.3355578

1 INTRODUCTION

For many Internet applications, especially Web and video applications, the underlying network is a key performance bottleneck. Poor networks cause large delays [41, 65] and require multiple round trips for object transfer [67, 71]. Since Web and Video account for over 70% of Internet traffic [69], there have been considerable advances in new network protocols, most notably HTTP/2 [38] for the Web, and DASH [22] for video applications.

IMC '19, October 21–23, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6948-0/19/10…$15.00
https://doi.org/10.1145/3355369.3355578

The key problem is that an application developer cannot easily choose the best protocol to use. For example, prior work has shown that for a Web application, the choice between using HTTP/2 and HTTP/1.1 is not straightforward [81]; Similarly, the DASH [22] protocol is used by a video player to fetch the video segment with the maximum bitrate, given the network throughput. However, as shown by the spate of recent works [6, 44, 72, 83, 85], predicting the network throughput, and in-turn choosing the right bitrate for the video segment is non-trivial.

Empirically evaluating all possible protocol and parameter 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. What is needed is an accurate application performance model to predict the right network protocol or parameter to use. Unfortunately, this is challenging because:

- Network performance is unpredictable [39], making it difficult to model higher layer application performance which depends on the network. Further, end-points do not have enough visibility into the dynamic network congestion at intermediate router buffers due to cross-traffic.
- The application layer interacts with the transport layer in non-intuitive ways [81]. Consequently, application performance depends not only on the application protocol, but also on the transport protocol, *and* their interactions.

In this work, we present ECON, a model that accurately and scalably predicts application layer performance. We show how ECON can be used by Web and Video applications to decide the best protocol or parameters at runtime. Since the performance of higher-layer applications depends on the performance of TCP [81], the core idea in ECON is to first model the underlying TCP and then use this as the building block for modeling application performance.

The key insight in ECON's TCP model is to combine analytical modeling with empirical data. Related works that model TCP fall into two categories: analytical or history-based. Analytical models make assumptions [17, 20, 24, 63] about packet loss that do not always hold true and hurts model accuracy (see §5). On the other hand, existing history-based models [34, 44, 72] predict throughput empirically based on historical data alone, ignoring the effect of TCP. History-based throughput predictions are inaccurate (see §5) because the network throughput is, in fact, regulated by the TCP congestion control algorithm.

ECON first builds an analytical model based on the congestion control algorithm employed by the underlying TCP. However, rather

than make assumptions about network losses, ECON drives the analytical model with *real-time empirical measurements*[1]. The empirical measurements are made *at the end points*, which serve as a proxy for inferring end-to-end network congestion without requiring explicit information about the intermediate traffic.

In this paper, we derive the model for TCP Cubic [32], which is the default TCP variant on Linux and Mac OS, and also show how our model can be extended to TCP Reno [2]. Because ECON explicitly takes into account the *slow-start phase*, it can also predict TCP performance for short flows. Note that our goal is to model network performance under *existing* choices to empower practitioners, and not to develop new TCP or HTTP variants.

We build on ECON's TCP throughput model to predict latency as well as the performance of HTTP/1.1 and HTTP/2. To this end, ECON extends the continuous flow model derived for TCP to *finite flows*. In the case of HTTP, since ECON models the performance of 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 evaluate ECON's TCP model with over 600 hours worth of experiments spread over several months across (1) four 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 [32] and Reno [2]). We compare ECON with seven alternative models — three analytical models and four history-based models (including those employed in the last few years [72, 83]). Our results consistently show that our modeling error is substantially lower than other models, often by more than 60%, even under dynamic network conditions. Our median throughput modeling error is *6%–16%* across all experiments.

ECON also accurately predicts the latency under HTTP/1.1 and HTTP/2 with a mean error of 3% and 7.4%, respectively. By predicting the workload and network conditions under which HTTP/2 outperforms HTTP/1.1 and vice-versa (§6), ECON allows practitioners to quickly decide which option to employ. Importantly, ECON makes these predictions without requiring extensive experimentation, unlike prior work [81].

To demonstrate the practical end-to-end applicability of ECON, we show (i) a Web server can leverage ECON to choose between HTTP/1.1 and HTTP/2 to improve the Web page load time, and (ii) how a video server (or client) can choose the optimal bitrate using ECON. In both cases, the application server collects empirical data for the connection to the client to build the model. The server then scalably predicts application performance using ECON for different protocols/parameters and helps pick the best option to employ.

For Web page loads, we show that, by taking the page load dependency structure into account, ECON improves page load time by 36%–56%, across different network conditions and Web pages. Likewise, for video applications, ECON's throughput prediction allows us to pick the optimal bitrate in most cases across different network conditions.

[1]Hence the name ECON model, that stands for *Empirically-augmented COngestion-aware Network model.*

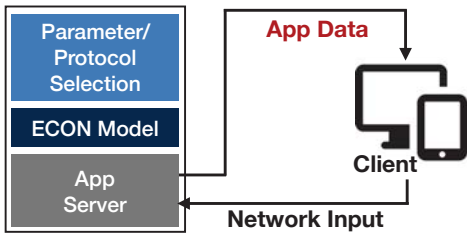


Figure 1: Illustration of the client-server environment that ECON operates in.

2 OVERVIEW AND MOTIVATION

To provide the context and scope of our work, we first discuss the network setup and envisioned use cases for ECON. We then provide the necessary background on TCP and highlight the need for ECON in light of existing works.

2.1 Target Network Environment for ECON

We envision ECON to be used in a client-server set up, where the client initiates the request but most data is served from the server, as shown in Figure 1. ECON predicts the network performance for the client-server pair by empirically analyzing existing network conditions on the server side. This empirical data is then used to drive the ECON model predictions at runtime. Note that there is nothing inherently server-specific in ECON; we choose to analyze the network conditions on the server side since the server performs the bulk of the networking in our set up.

ECON does not make any assumptions about the location of the bottleneck along the server-client link. Instead, by monitoring the performance at the end-point, ECON infers the network conditions on the server-client link. If there is no existing flow between the server and the target client, ECON can obtain estimates of the parameters based on the server's existing or recent connections to other clients in the geographical vicinity of the target client, similar to recent works [42, 72].

We show how ECON can be extended for different applications. For example, for the adaptive video streaming use case in §7, ECON predicts the performance under different bitrates and informs the server about the highest bitrate that can be currently used to deliver seamless video streaming to the client. This information can also be relayed to the client, in case the client-side video player needs to make the bitrate decision. Similarly, for the Web use case in §6.4, ECON predicts the web page load time under HTTP/1.1 and HTTP/2, and relays the optimal decision to the client.

2.2 Background on TCP

Transmission Control Protocol or TCP is the widely used protocol in the transport layer [27]. 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. cwnd is in the unit of packets.

TCP regulates cwnd using a congestion control algorithm. We discuss loss-based algorithms here and discuss other variants in §8.

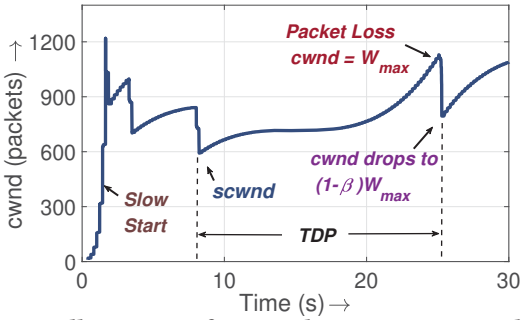


Figure 2: Illustration of TCP Cubic congestion window size (cwnd) evolution over time.

A loss-based TCP protocol 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, TCP increases cwnd less aggressively, until there is a loss. The exact cwnd regulation depends on the TCP variant employed, but the rise and fall of cwnd is typically periodic.

Figure 2 shows the evolution of cwnd under TCP Cubic [32], the most popular TCP variant. The figure shows cwnd in terms of RTTs for the *Azure* testbed. After slow-start, TCP increases the cwnd via the equation:

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

where C is a constant, $K = \sqrt[3]{\frac{W_{max}\beta}{C}}$, t is the time 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 the slow start phase.

Some TCP variants regulate cwnd using the Additive Increase, Multiplicative Decrease (AIMD) algorithm. This algorithm increases cwnd linearly in each round. But when a loss occurs, it rapidly reduces cwnd using a multiplicative factor. For example, TCP Reno [2] increments cwnd by 1 in each RTT, and drops cwnd by a factor of 2 upon a loss.

2.3 Existing TCP models

Researchers have developed analytical [10, 17, 20, 24, 59, 63], and data-driven [34, 44, 72, 83] models to characterize TCP throughput (and/or latency) as a function of various network parameters. Both sets of models have certain shortcomings, as we discuss next.

Analytical TCP models typically work by tracking the number of packets sent in each RTT, depending on the cwnd evolution, as dictated by the underlying congestion control algorithm. The above-cited analytical models only focus on TCP Reno [2], and all but one model focus only on the congestion avoidance phase. Further, such analytical models typically make simplifying assumptions that do not always hold in practice, such as: (1) assuming that the *loss probability can be modeled either as a constant or as some known distribution*, e.g., Poisson. Figure 3 shows the empirical relationship between loss probability and cwnd for one of our real-world testbeds. Clearly, loss probability is not a constant, and also depends on the number of parallel connections employed; this is important when comparing HTTP/1.1, which uses parallel TCP

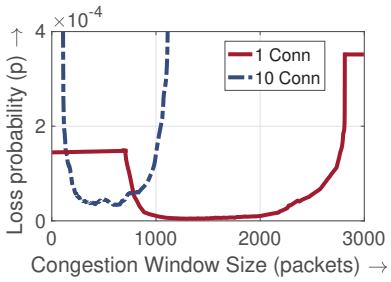


Figure 3: Empirical results for loss probability (p) vs. congestion window size (cwnd) for the *Azure* network under TCP Cubic for different number of parallel connections.

connections, and HTTP/2, which only uses a single TCP connection; (2) assuming that the *starting congestion window sizes are identically distributed*. However, starting congestion window size varies across flows, especially when TCP connections are reused. While the effect of starting congestion window is amortized for bulk transfers, the impact is significant for short transfers; and (3) focusing on *static network conditions where the parameter values, such as loss probability, are assumed to be stable*. However, *network conditions can be very dynamic in practice* [19, 34, 40, 78]. In fact, for video streaming, today's media players routinely employ adaptive bitrate streaming to adjust to variations in network conditions [42, 72].

History-based, data-driven models, on the other hand, leverage historical observations of network performance to predict future performance. However, by relying only on historical data, such models limit their responsiveness. For example, when the RTT is small, the congestion control algorithm results in a very dynamic cwnd, and consequently throughput evolution, which is hard to predict using historical observations alone. Further, historical observations might not be accurate predictors for future performance as they do not account for dynamic events, such as a very recent packet loss (which will result in an abrupt drop in throughput), or anticipated events, such as an imminent loss due to the addition of new connections or an inflated cwnd.

The above shortcomings of analytical and data-driven models hurt the model accuracy in practice, 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. 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 2 shows the evolution of the congestion window size, cwnd (in packets), under TCP Cubic [32] during the slow-start and congestion-avoidance phases. We model the throughput for these phases separately.

ECON leverages the periodicity of the cwnd behavior at steady state to predict throughput of the current triple duplicate period

(TDP). The TDP is defined as the period between a starting congestion window and the first loss, indicated by triple duplicate ACKs. During this period, the cwnd 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, using renewal reward theory [21]. For example, starting from a starting congestion window size of scwnd packets in Figure 2, the throughput is the number of packets sent between scwnd and W_{max} divided by 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 probability is constant. In fact, as indicated in Figure 3, the loss probability will change at each point in the TDP curve because the loss rate depends on the cwnd. To this end, ECON takes as input (i) the empirical relationship between loss probability and cwnd, and (ii) the starting congestion window value, scwnd, to estimate the expected number of packets sent before the loss occurs. We next describe how we obtain the relationship between loss probability and cwnd.

3.2 Relationship between loss probability, p, and congestion window size, cwnd

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 are 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 total number of losses recorded at cwnd divided by the total packets sent at that cwnd, over the monitoring period. Some cwnd values may not be encountered during a monitoring period; in that case, we set the loss probability for this cwnd to that of the closest cwnd where loss is encountered.

In detail, the process of deriving p(cwnd) is as follows. The server maintains: (1) a hashmap A which records the frequency of each cwnd value encountered when sending a packet; and (2) a hashmap B which records the frequency of the cwnd value encountered when a loss occurs. By dividing the values in B by A, we get p(cwnd). The p(cwnd) function consumes about 5KB in memory.

We infer p(cwnd) by observing losses at the end point, e.g., the server, so the effect of cross-traffic at intermediate routers is implicitly taken into account. We also do not need to explicitly model the effect of queue management techniques, such as RED [13, 60].

p(cwnd) relationship for real-world networks: Figure 3 shows the empirically obtained p(cwnd) curve based on one hour of monitoring for *Azure* network under TCP Cubic (more details in §5). Note that the measurements are between Azure VMs so the traffic likely does not leave Azure's private backbone to take public Internet routes. For 1 connection (red curve), we did not encounter losses before cwnd≈800 and after cwnd≈2800, so we instead use the closest non-zero loss probabilities for these points, resulting in the flat horizontal line segments.

Clearly, the p(cwnd) relationship is *not* monotonic. In general, the loss rate initially decreases with an increase in cwnd (likely because a smaller cwnd suggests poor network conditions), but then the relationship is reversed due to congestion.

TDP	triple duplicate period
$cwnd$	congestion window size
$scwnd$	starting congestion window size
npc	number of parallel connections
N	number of packets sent in a TDP
M	number of packets before loss is detected
X	number 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.

The p(cwnd) relationship also depends on the number of parallel connections (npc), as shown in Figure 3, since the bandwidth is now shared between more connections. We typically observe smaller cwnd values for higher number of parallel connections, due to the increased congestion experienced by each connection. In our experiments, we capture this empirical relationship, p(cwnd, npc), by attributing losses for a given connection to the specific cwnd *and* the specific npc value at which the loss occurs. While a more accurate p(cwnd, npc) model can be developed by also looking at the cwnd value of the other (npc - 1) connections, the resulting model will be quite complex. We find that our simpler model suffices for accurate predictions.

We obtain similar curves for the other network setups in our experiments, including those over WiFi. We have made all the p(cwnd) experimental data, for all networks, publicly available on a Github repository [26]. We find that the temporal stability of the p(cwnd) function depends on the network type. Generally, the p(cwnd) function is more stable in cloud networks (the *Azure* network, in our case, see §5.1) and less stable in the wild.

3.3 Modeling TCP Cubic throughput

We now provide an overview of our throughput modeling approach for TCP Cubic, which is the most widely used TCP variant (the default version on Mac OS and Linux). We will then discuss our model extension for TCP Reno. We defer the exact mathematical derivations to Appendix A. The model parameters and random variables we employ in our analysis are listed in Table 1 for reference.

ECON takes as input the empirical p(cwnd, npc) relationship and the starting congestion window size, scwnd. Then, the throughput B is given by:

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

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 depend 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; then, (N-M) is 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 that the first 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 $Pr(M = 2) = (1 - p(scwnd)) \cdot p(scwnd)$, since p(cwnd) is the loss probability for any packet in the congestion window of size cwnd. 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, p, since p depends on cwnd. In general, if the loss happens at the m^{th} packet of the n^{th} congestion window, we have

$$M = \left(\sum_{j=1}^{n-1} cwnd_j\right) + m, \text{ and}$$

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

where $cwnd_j$ is the cwnd in the j^{th} RTT, given by $cwnd_j = C((j - 1) \cdot RTT - K)^3 + scwnd/(1-\beta)$, via Eq. (1); note that $cwnd_1 = scwnd$. We then obtain $E[M]$ by conditioning over M and generalizing to parallel connections (see Appendix A).

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:

$$E[N] = (E[M] - 1) + cwnd_{E[X]}, \quad (3)$$

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. Once E[X] is derived, we can easily obtain E[N] using Eq. (3).

Finally, we obtain the throughput B according to Eq. (2); see Appendix A for details. In §4.1 we describe how we estimate the bandwidth in practice based on the above modeling. Note that ECON's model is not in closed-form, thus limiting our ability to analyze the throughput model via simple visual inspection of the final expression. Also, ECON leverages more available information than alternative analytical models, like PFTK. However, by using empirical data, ECON enables much more accurate network performance predictions, as we show in §5.

3.4 Extension to TCP Reno

The above model can be extended to other loss-based TCP variants; we now 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. Specifically, the cwnd at the j^{th} RTT is $cwnd_j = scwnd + j - 1$.

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 is similar to Cubic, and is thus omitted. The key difference when modeling the throughput under TCP Reno is 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, $E[B_s]$, is a function of (i) the number of packets sent in slow-start before a loss occurs or the slow start threshold is reached, N_s, and (ii) the evolution of cwnd during slow start:

$$E[B_s] = E[N_s]/((E[X_s] + 1) \times RTT), \quad (4)$$

where 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). 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 $p(icwnd \times 2^{k-1}, npc)$. We use this loss probability to derive N_s and X_s, and consequently B_s, via Eq. (4). The modeling for slow start proceeds similarly as for Cubic, and is thus omitted.

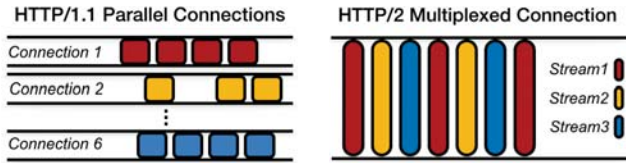
4 ECON

The TCP modeling is the building block over which we build ECON's application layer prediction. In this section we discuss how ECON is used in practice and then describe the application-layer models.

4.1 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. Such a bootstrapping step is required for all history-based prediction models [34, 44, 72, 83]. The p(cwnd) relationship can be obtained from an ongoing or recent flow between the server and client. Alternatively, the p(cwnd) function can be computed for the few seconds of the object transfer and then used for the remaining transfer, as in the case of long-running flows such as video. If neither of these work, e.g., for short-running flows, then we can leverage the clustering technique used in recent work [42, 72] as follows. For cases where the p(cwnd) function needs to be computed at a server that connects to several thousands of clients, the idea is to cluster connections based on the geographical location of the clients and other features. The p(cwnd) relationship for a new client can then be estimated as the median or mean p(cwnd) of other connections in the cluster that the client is closest to.

In practice, the p(cwnd) relationship can change dynamically for several reasons, such as failures, rerouting along the path, or increased traffic due to colocated flows on the path [9, 47]. We use a "sliding window" approach to monitor network conditions periodically, including RTT, losses, and the number of packets sent at each congestion window, over a certain sliding window size. We then periodically update the model parameters, including the p(cwnd, npc) function.



(a) HTTP/1.1 creates parallel TCP connections.

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

Figure 4: HTTP/1.1 versus HTTP/2.

Using the empirical measurements, we estimate expected throughput using Eq. (2). Note the infinite summation over cwnd values. In practice, we rarely see cwnd values greater than a few thousands, so we cap the summation.

4.2 Application modeling: Latency

Our first step towards application-layer modeling is to model the latency of a TCP flow. So far, we modeled the throughput, B. Consider data of size x to be transmitted at time t on a single connection. Latency is the time taken to transmit data of size x, and we estimate this latency as a function of B:

$$L = x \cdot RTT/(B \cdot MSS), \tag{5}$$

where RTT is the Round Trip Time and MSS is the Maximum Segment Size.

Recall that the throughput B is a function of scwnd and the p(cwnd) curve. If the data transfer is part of an existing flow, then we set scwnd to be the cwnd at that time t, $cwnd_t$. We use the p(cwnd) relationship of the existing flow, and use Eq. (2) to estimate the throughput. If the data transfer is not part of an existing flow, we use our slow-start model (§3.5) and set scwnd to be the initial cwnd set by TCP. The p(cwnd) curve is obtained using the bootstrapping step described in §4.1. Finally, the throughput is estimated via Eq. (4).

We next extend the latency estimation to multiple connections in the context of HTTP.

4.3 Application modeling: HTTP

Both HTTP/1.1 and HTTP/2 work over TCP at the transport layer. In both cases, the client sends an HTTP request for an object and the server sends an HTTP response with the requested object. Figure 4(a) shows an example HTTP/1.1 persistent connection; HTTP/1.1 creates a TCP connection for each object. Once the object is received, the client requests the next object. HTTP/1.1 can create parallel TCP connections to fetch objects simultaneously. Most browsers limit the number of parallel connections per server to six [84].

The difference between HTTP/1.1 and HTTP/2 is in how the TCP connections are leveraged. Figure 4(b) shows an example HTTP/2 connection. In HTTP/2, the client creates a single TCP connection and requests multiple objects on this single connection. The multiplexed requests are called *streams*, and HTTP/2 has a limit on the number of streams [11].

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).

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 reduce the number of parallel connections and update the 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. We 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 replaces a completed stream in HTTP/2.

5 ECON TCP MODEL EVALUATION

We evaluate our TCP throughput and latency modeling accuracy under four different real world networks for: (i) TCP Cubic, (ii) TCP Reno, and (iii) a WiFi network.

5.1 Methodology

Networks and experimental setup. We conduct experiments on four real-world networks:

(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 (VMs) hosted in the Japan East, East US 2, West US 2, Central US, and South Central US locations of Azure [57]. The average RTT within *Azure* is 10–200ms depending on the receiver location. Unless specified otherwise, we report results for the receiver in Japan East (RTT ≈ 200ms).

(2) **Southeast:** The receiver (VM) is located in a CloudLab site at Clemson University and the sender (bare-metal) is located at Stony Brook University with average RTT of 23ms.

(3) **Northeast:** The sender (bare-metal) and the receiver (bare-metal) are both located at Stony Brook University with an average RTT of 50ms.

(4) **Long-distance:** This is a collection of networks with the sender (bare-metal) hosted at Stony Brook University and the receivers located at University of Washington and Korea, resulting in average RTTs of 60–200ms. Unlike the above networks that are partially part of a cloud data center, the *Long-distance* network is, to the best of our knowledge, over commodity links.

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 prediction error is not affected much by the SACK and Delayed ACK settings. The only parameters that we do change are the Linux TCP read and write buffer sizes, 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 TCP buffers. For the two cloud-based networks, *Azure* and

Southeast, we observe relatively low loss rates. We thus use TC [1] on the sender side to experiment with different loss probabilities for these networks.

Data collection. All senders and receivers run the Ubuntu OS (v14.04.5 or later). We use *iPerf* [76] to send TCP traffic from sender to receiver. For each network, we run multiple 1-hour experiments across several days over a period of 12 months with varying number of TCP connections (from 1 to 10); in total, we have about 660 hours worth of experimental data. Each experiment begins with the slow-start phase.

We use the Linux TCP probe module [3] 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 three formula-based (FB), or analytical models, and four history-based (HB), or data-driven models.

- The classic **PFTK** model [63].
- The model proposed by Chen et al. [20], 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 Reno's throughput by accounting for fast retransmit/fast recovery and slow start.
- Last Sample (**Sun16-LS**) [72], a simple History-Based (HB) model that predicts throughput based on the most recently observed throughput value.
- Exponentially weighted moving average (**He05-EWMA**) [34], a parameterized HB model that predicts throughput based on previously observed throughput values.
- Holt-Winters (**He05-HoltWinters**), also a parameterized HB model that is well suited to non-stationary processes, and predicts throughput by capturing the trend in previously observed values, as detailed in He at al. [34].
- Harmonic Mean (**Yin15-HM**) [44, 83], a HB model that predicts throughput based on the harmonic mean of past observed throughput values.

The PFTK model [63] was published before 2004, when a draft of the Cubic variant was first introduced [66]; as such, the first three (PFTK-based) models only apply to TCP Reno. We thus compare our Cubic model with the other four HB models for evaluation. For TCP Reno model evaluation, we compare with all seven alternative models.

Sensitivity analysis. For a fair comparison, we enhance the three analytical models [20, 24, 63] to use the same sliding window approach as our model (§4.1). The four HB models 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 the sliding window size between 10s and 100s; values above 100s result in stale data and values less than 10s result in high computation overhead. For formula-based models (PFTK, Chen06, Dunaytsev07, and ECON), we find that a large window size works better, so we set the window size to 100s for these models. For HB models, a smaller sliding window works well, so we set it to 10s. For all models, an update frequency of 10s works well.

The He05-EWMA and He05-HoltWinters also have two additional parameters, α and β, that need tuning. Our sensitivity analysis showed that $\alpha = 0.8$ and $\beta = 0.8$ work best for our testbed, so we use these parameter values.

5.2 Cubic & Reno throughput prediction

We now evaluate the throughput prediction error for TCP Cubic and Reno. We predict the throughput for the next 1s.

CDF of TCP Cubic throughput prediction errors: Figure 5 shows the CDF of Cubic throughput prediction errors for all models and all networks. These results are based on 22,000 prediction windows across all networks. We compute the prediction error by comparing our predictions with the observed experimental values.

Our model *significantly outperforms* all other models for all networks. Our median throughput prediction error for *Azure*, *Southeast*, *Northeast* and *Long-distance* is about 11.5%, 15.5%, 9.2%, and 11.6%, respectively. By comparison, the best median throughout prediction error for alternative models is 36.1%, 67.8%, 28.5% and 33.5%, for the four networks. In terms of average error, ECON reduces the relative prediction error compared to the best alternative model by 85.7%, 92.1%, 72.2% and 64.1%, for the four networks, respectively.

In general, we find that for networks with small RTTs, Sun16-LS has the lowest errors and He05-HoltWinters has the largest errors, among alternative models. Note that in Figure 5(b), for the *Southeast* network, the prediction error is large for alternative history-based models. This is because this network has a small RTT and so Cubic's congestion control results in a very dynamic cwnd evolution, which is hard to capture using historical observations alone.

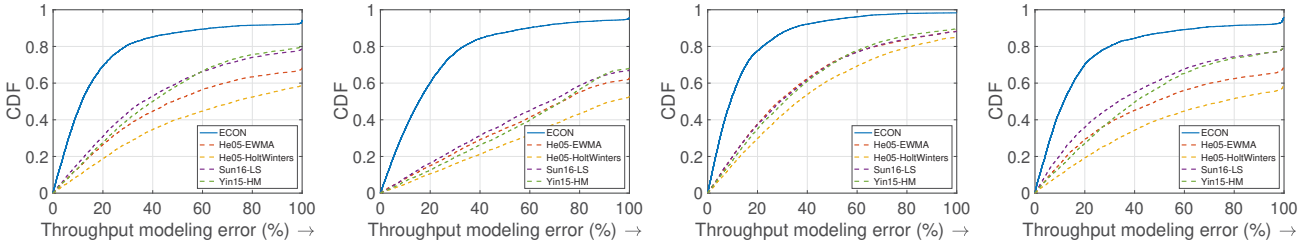
To investigate whether the TCP read and write buffer sizes affect our prediction accuracy, we also evaluate our predictions for TCP Cubic when using the system default TCP buffer sizes (4MB for write and 6MB for read) in the *Northeast* network. Our median throughput prediction error in this case is 9.6%, whereas the best median throughput prediction error for alternative models is 34.9%.

CDF of TCP Reno throughput prediction errors: We evaluate the prediction error for TCP Reno over all four networks and compare with all seven alternative models as they apply to TCP Reno.

Figure 6 shows the CDF of TCP Reno throughput prediction error for all networks. These results are based on 44,000 prediction windows across all networks. As before, ECON *significantly outperforms* all seven alternative models. Our median throughput prediction error for *Azure*, *Southeast*, *Northeast* and *Long-distance* is about 6.1%, 11.3%, 10.6%, and 8.6%, respectively. By comparison, the best median throughout prediction error for alternative models is 23.4%, 22.7%, 20.1% and 24.8%, for the four networks.

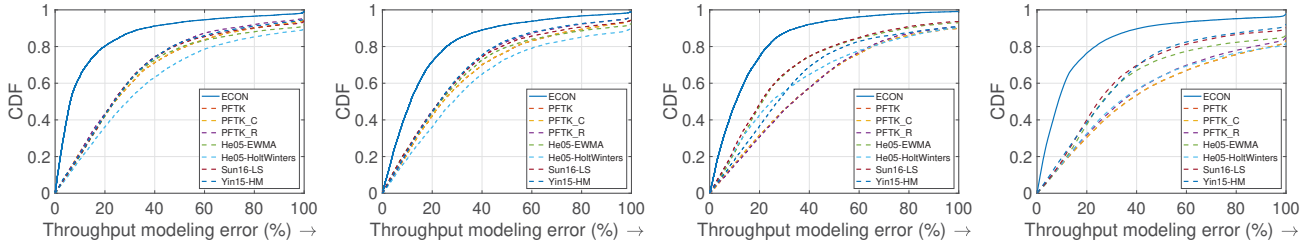
Sensitivity to model parameters: We also experiment with a smaller sliding window (training set) size of 10s for all models. Results are qualitatively similar, with ECON improving the throughput prediction error compared to the best alternative model by 48% and 39%, respectively, for Cubic and Reno.

Finally, in addition to the 1s prediction window used above, we also evaluate ECON's performance under a 5s and 10s window. Again, ECON outperforms all alternative models we experiment with. The average reduction in prediction error afforded by ECON over the



(a) Azure network. (b) Southeast network. (c) Northeast network. (d) Long-distance network.

Figure 5: CDF of TCP Cubic throughput prediction (for the next 1s) error under all eligible models and under all networks.



(a) Azure network. (b) Southeast network. (c) Northeast network. (d) Long-distance network.

Figure 6: CDF of TCP Reno throughput prediction error under all models and under all networks.

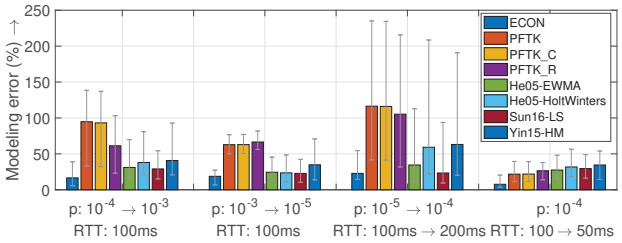


Figure 7: Throughput prediction error under changing network conditions. ECON outperforms other models.

Network	ECON	EWMA	H.Winters	LS	HM
Azure	10.4	35.5	56.4	29.6	31.5
SouthEast	13.1	59.0	83.0	59.6	63.9
NorthEast	8.7	25.4	34.1	26.5	28.1
Long-distance	12.5	34.3	59.2	26.7	34.3

Table 2: Median error (in %) for Cubic latency modeling.

best alternative model is about 50% and 38% for Cubic and Reno, respectively, under 5s prediction, and about 43% and 23% under 10s prediction.

Analyzing model efficacy for different scwnd and npc settings: We find that ECON outperforms other models for all starting congestion window (scwnd) values. However, the improvement is more pronounced for smaller and larger scwnd values. For example, under the *Northeast* network, for $scwnd < 50$, ECON lowers the median throughput prediction error range from 158–264% under the best alternative model (Yin15-HM) to about 15%. For $scwnd > 700$, ECON lowers the median throughput prediction error range from 58–61% under the best alternative model (He05-HoltWinters) to about 21%. Results are similar for other networks.

This is because the alternative 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 3).

We also evaluate ECON by analyzing its prediction accuracy for different number of parallel connections (npc). Our model outperforms other models irrespective of the number of parallel connections; further, our median error numbers are much lower, by about 41–76%, compared to the corresponding numbers for other models.

Analyzing the effectiveness of the sliding window approach: To evaluate the dynamic nature of ECON, we consider the *Northeast* network running TCP Reno. We use TC [1] to change the RTT and loss probability, p, to emulate varying network conditions.

We run a 10min experiment starting at $t = 0min$ with a 100ms RTT and $p = 10^{-4}$. Then, every 2 minutes, we change the network conditions as follows:
$p = 10^{-4} \rightarrow 10^{-3}$ at $t = 2min$,
$p = 10^{-3} \rightarrow 10^{-5}$ at $t = 4min$,
$p = 10^{-5} \rightarrow 10^{-4}$ and $RTT = 100ms \rightarrow 200ms$ at $t = 6min$,
$RTT = 200ms \rightarrow 50ms$ at $t = 8min$.
The experiment is stopped at $t = 10min$. As before, we predict the throughput for the next 1s.

Figure 7 shows the median prediction error for each of the four 2-min intervals after a change in network conditions. We compare with all other TCP Reno modeling alternatives. Overall, ECON results in much lower prediction error, by about 56% on average, compared to the other models across all four dynamic cases. Compared to the best alternative model for each case in Figure 7, ECON improves the median prediction error by about 32%, on average.

5.3 Latency prediction errors

An immediate application of our TCP throughput model is predicting the latency of TCP flows. Table 2 shows the median latency prediction error under TCP Cubic for all networks and all eligible

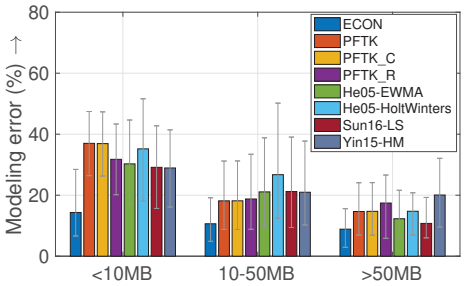


Figure 8: Latency prediction error for different data size ranges. ECON provides greater benefits for smaller data sizes.

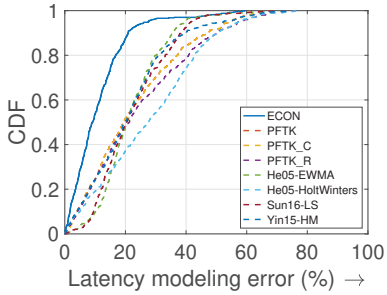


Figure 9: TCP Reno latency prediction error for a wireless connection in the *Southeast* network.

models. These results are based on more than 5000 data transfer experiments, with the data size in each transfer ranging from 1MB to 100MB. While some of the alternative models, such as He05-EWMA [34], only focus on throughput modeling, we extend their models to also predict latency, similar to our model.

Our model consistently outperforms all other models for *all* networks. The median latency prediction error of ECON under Cubic is about 11%, whereas that of other models is about 43%. Results are similar under TCP Reno, with the median prediction error across all networks under ECON being around 10%, while that of other models ranges from about 25% (for Sun16-LS) to 30% (for He05-HoltWinters).

Figure 8 shows the median prediction error under TCP Reno for the *Southeast* network 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 median latency prediction error for ECON for the three ranges (<10MB, 10–50MB, >50MB) is 14.3%, 10.6%, and 8.9%, respectively. By contrast, the best alternative model has median error of 28.9%, 18.2%, and 10.8%, respectively. The improvement afforded by our model over other models roughly 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, (ii) unlike alternative models, we leverage the scwnd value in ECON and find that throughput prediction is more sensitive to scwnd at short timescales, and (iii) other analytical models, such as PFTK, use steady-state analysis with constant loss rates, which is well suited for bulk transfers but not short flows [63]. By contrast, ECON performs well for short transfers as well.

5.4 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 use 1 TCP connection and run experiments for 5 hours.

Figure 9 shows the CDF of the latency prediction error under TCP Reno (so we can compare with all alternative models); results are similar under Cubic and for throughput modeling. Compared to the best alternative model, ECON reduces the median prediction error from 19.3% (PFTK) to 9.1%, and average error from 21.4% (Yin15-HM) to 11.2%.

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

Although we do not explicitly model RTOs (retransmission timeouts), ECON still achieves high accuracy when RTOs are present. In our wireless experiments, for example, the loss probability due to RTOs is around 10^{-7}. Despite these RTOs, ECON's average prediction error is only about 11% for the wireless testbed.

5.5 Modeling results for emulated last mile networks

In this subsection, we evaluate ECON in a more realistic network setting. Specifically, we emulate last mile networks in the *Northeast* network with more representative bandwidth, RTT, and loss rates using TC. We consider regimes with loss rates of 0.1% to 1%, RTT of 30ms, and bandwidth of 50Mbps. These chosen parameter values are based on the public 4G-LTE data of T-Mobile [74] and Verizon [79].

We evaluate 3 different loss rates in the *Northeast* network: 0.1%, 0.5% and 1%. For each loss rate, we run an iPerf3 experiment for 1 hour. Figure 3 shows the median throughput prediction error for each loss rate under ECON and other TCP Cubic modeling alternatives. As before, ECON achieves much lower throughput prediction error when compared to the best alternative model in all cases.

Loss Rate	ECON	EWMA	H.Winters	LS	HM
0.1%	7.8	25.5	31.3	25.3	26.9
0.5%	15.3	21.9	24.6	22.5	20.9
1%	18.5	21.5	22.9	21.4	19.9

Table 3: Median error (in %) for Cubic in emulated last mile networks (bandwidth=50Mbps, RTT=30ms).

6 WEB: HTTP/1.1 AND HTTP/2

We now (1) evaluate ECON's latency prediction under HTTP/1.1 and HTTP/2, and (2) use ECON to help a Web application choose between HTTP/1.1 and HTTP/2 for Web page loads.

6.1 Experimental setup

We evaluate ECON's HTTP models by sending HTTP traffic between a browser and a Web server in the *Northeast* network. We use the Chrome browser (v63.0.3239.132) as client 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.

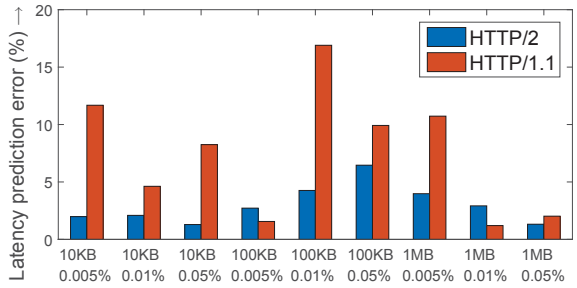


Figure 10: HTTP/1.1 and HTTP/2 latency modeling error for different file sizes and loss rates. For the first three sets of bars, the transfers complete within the TCP slow start phase.

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.

6.2 HTTP prediction results

Figure 10 shows our latency modeling error for HTTP/1.1 and HTTP/2 across 9 different experiments; we use ECON's HTTP models from §4.3 to predict the latency of transferring objects. 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 3.0% and 7.4%, respectively.

When fetching 10KB objects, both HTTP/1.1 and HTTP/2 take about 50-150ms during which TCP remains in the slow-start phase. This result shows that ECON can accurately predict performance even when TCP is in slow start phase.

6.3 Predicting HTTP/1.1 versus HTTP/2

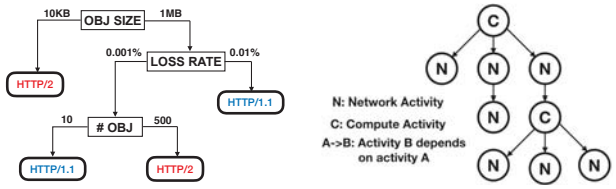
We now return to the key question we posed earlier—should we use HTTP/1.1 or HTTP/2 for a given environment and workload. We explore 16 different scenarios of workload and network conditions by varying the object size (10KB, 1MB), loss probability (10^{-5}, 10^{-4}), number of objects (10, 500), and RTT (50ms, 200ms). For each scenario, we predict the latency under HTTP/1.1 and HTTP/2 using our models.

Figure 11(a) illustrates our prediction results as a tree diagram that determines when HTTP/1.1 outperforms HTTP/2 and vice-versa. 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. For all 16 scenarios, our results are in agreement with the empirical results obtained by prior work [81], but without requiring extensive experimentation.

Importantly, making the wrong choice between HTTP/2 and HTTP/1.1 can increase latency by up to 4.6×, highlighting the importance of accurate models. The average increase in latency, across all 16 scenarios, when making the wrong choice, is about 264%.

6.4 Extending ECON to Web

We have shown above that making the right choice between HTTP/1.1 and HTTP/2 is critical to performance. The next question is, how



(a) ECON prediction results for HTTP/1.1 vs. HTTP/2.

(b) An example dependency graph for a Web page load.

Figure 11: Decision tree and Web dependency graph.

can a Web application use ECON's prediction to choose between the two HTTP protocol choices.

Extending ECON's prediction to the Web is challenging because of the complexity of Web page loads. A Web page embeds tens of objects of varying sizes. The objects are not all downloaded at the same time, and the timing depends on other compute activities and the dependencies between the objects [81]. This means that the impact of using HTTP/1.1 and HTTP/2 for page load time is more nuanced.

6.4.1 Use case in practice. The page load process starts when the client browser requests the URL from the Web server. The server parses the URL request and determines the set of objects that must be fetched to load the Web page. This can be done by loading the page prior to the request, ignoring personalization. The server also computes the p(cwnd) relationship for the connection, employing bootstrapping, if needed, as discussed in §4.1 and in prior works [42, 72].

Based on the set of objects in the Web page, the network information, and the starting congestion window, the server runs an emulation algorithm (discussed below) to predict the latency for the page load under HTTP/1.1 and HTTP/2. The latency-minimizing protocol choice is then relayed by the server to the client, along with the requested HTML file. Our evaluations show that making this decision incurs insignificant latency and does not affect the page load process.

6.4.2 Predicting the Web page load time (PLT). Our general idea is to deconstruct the page load process into object requests and reconstruct it using ECON modeling, based on the protocol to be used (HTTP/1.1 versus HTTP/2) and the underlying network parameters. Reconstructing the page load is tricky because of dependencies in the page load process and the interspersed object load and compute activities.

To account for dependencies between Web activities, the server first determines these dependencies using tools such as WProfX [82]. Figure 11(b) illustrates the dependency graph for a simple example Web page, showing the network activities (N), compute activities (C), and dependencies.

The emulation algorithm proceeds as follows: for the first set of network object loads, we predict the latency of fetching the objects under HTTP/1.1 and HTTP/2. We then shift the dependency graph based on this latency as needed. We then calculate the time to transfer the second set of objects depending on the first set of network object loads, and so on. Note that such an emulation algorithm is needed for Web page loads that have complex dependencies between objects, since we cannot simply divide the page size by the

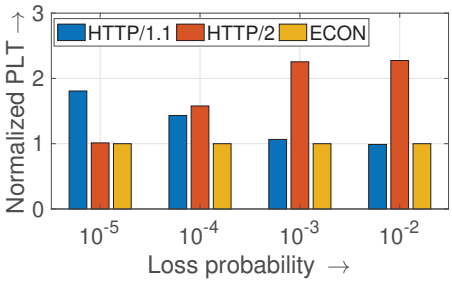


Figure 12: The average page load time of HTTP/1.1, HTTP/2, and ECON under different loss probabilities.

throughput to get the page load time. If the next level is a compute activity, we assume it takes a constant time, based on prior runs. We continue this process until all activities are completed. We use this completion time as our predicted page load time, PLT, which is a common Web performance metric [14, 81].

6.4.3 Evaluation methodology. For evaluation, we choose 10 Web pages from the HTTP Archive dataset [33] that exhibit a range of number of objects per page. Specifically, we look at pages that have 20 to 500 objects/page, and randomly choose 10 pages from this subset. The Web server is Nginx 1.10.3 and the client browser is Chromium 73.0.3679.0. We use record and replay to load the Web page from our local server [49].

We vary the loss probability to emulate 4 different network conditions: $p = 10^{-5}$; $p = 10^{-4}$; $p = 10^{-3}$; and $p = 10^{-2}$. In all cases, we set the RTT to 50ms. To obtain the ground truth, we load each page using HTTP/1.1 and HTTP/2 under each network condition 10 times and use the median.

Loss probability (p)	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Prediction accuracy	10/10	8/10	8/10	8/10
PLT improvement	44.8%	36.5%	55.6%	56.0%

Table 4: ECON prediction results, compared to the ground truth, for the choice between HTTP/1.1 and HTTP/2 for 10 Web pages loaded under different loss probabilities.

6.4.4 Evaluation results. Table 4 shows the results of ECON's prediction for the choice between HTTP/1.1 and HTTP/2 compared to the ground truth. For very low loss probabilities, ECON makes the right choice between HTTP/1.1 and HTTP/2 for all 10 Web pages. For other cases, ECON predicts the right protocol choice 80% of the time. The PLT penalty when making the wrong choice can be significant. We estimate the PLT when using ECON and when making the wrong choice between HTTP/1.1 and HTTP/2. Across different network conditions, ECON reduces PLT by 36%–56%, see Table 4.

Figure 12 shows the average PLT, across all pages, under different network conditions when using only HTTP/1.1 or only HTTP/2, and when using ECON; results are normalized by the PLT under ECON. Clearly, using only HTTP/1.1 or HTTP/2 hurts PLT significantly under certain conditions.

ECON runs on a bare-metal Linux server with an Intel i9-7900X CPU and 32GB of RAM. On average, for each Web page, the CPU runtime of ECON is only 0.8% of the PLT. The maximum memory overhead, when running the model, is only 0.4%.

7 IMPROVING ADAPTIVE VIDEO STREAMING VIA ECON PREDICTIONS

Video traffic is quickly becoming *the dominant traffic* on the Internet [23, 44]. Much of the video traffic today is delivered by HTTP-based protocols such as HTTP Live Steaming (HLS) [37] and Dynamic Adaptive Streaming over HTTP (DASH) [22]. Under HTTP-based video delivery, the content provider hosts segments of videos encoded at different bitrates. The higher the bitrate, the larger the segment size. With adaptive streaming, the client (video player) controls the video streaming quality by dynamically adjusting the bitrate to achieve the highest Quality-of-Experience (QoE) [6]. The video player aims to maximize the quality of the video (the bitrate served), while ensuring that the video segment can be downloaded without rebuffering; the download time depends on the network conditions. Since network conditions can change dynamically during video delivery, recent work has argued for *better network throughput predictors* to improve the quality of video streaming [72, 83, 85].

This section investigates the efficacy of ECON's network predictions in determining the next video segment's bitrate.

Experimental setup: For this case study, we consider a video hosted by a server that comprises a number of segments, each encoded at a different resolution, with higher resolution segments having larger file sizes. Each segment corresponds to about 3.1s of playback time; we denote this segment duration as T. We use Nginx [4] as our video server and VLC as our video player client. The video is streamed over the DASH protocol. We use the *Northeast* network under TCP Cubic, and vary network parameters via TC.

Methodology: We consider midstream adaptation [72], meaning that the video streaming is ongoing, and we wish to select the bitrate for the next segment. For this use case, we state the QoE goal as selecting the highest resolution (or bitrate) segment that can be transferred from the server to the client in less than T seconds (3.1s, in our case). We assume that the video has been streaming for a couple minutes, and so there is sufficient historical data for history-based models and for ECON to compute the p(cwnd) relationship.

In our setup, the bitrate selection can be managed by either the client or the server. While the video bitrate selection is usually driven by the client, some recent works have demonstrated the benefits of running ABR (adaptive bitrate streaming) algorithms on the server side [6, 42]. If the server is managing the bitrate adaptation, based on the ECON throughput model, the server decides on the highest resolution (and file size) segment that can be completely transferred in T secs. Alternatively, if the client is managing the bitrate, then the server still makes the decision about bitrate as discussed above and conveys the optimal resolution to the client. The client can then make the bitrate decision based on this server side notification.

Evaluation results: We consider networks with different RTT values under a high loss probability of 0.01 to evaluate the predictions under lossy networks. In all cases, we first establish the ground truth by playing the video under all supported resolutions (1080p, 720p, 480p, 360p) and identifying the highest resolution that allows segments to be transferred in less than T (segment duration) seconds. Then, using the various models, and ECON, we predict the

RTT	G Truth	ECON	EWMA	HW	LS	HM
50ms	720p	720p*	720p*	720p*	720p*	720p*
100ms	720p	720p*	720p*	360p	480p	720p*
150ms	480p	360p	720p	360p	720p	720p
200ms	480p	480p*	720p	720p	720p	720p

Table 5: Optimal resolution predicted by different models.

highest resolution for the next segment using the model-specific throughput predictions for the next T seconds, starting from a mid-stream position. Since video streaming under DASH uses a single connection without multiplexing [61, 75], we adopt the ECON Cubic model and other history-based alternatives.

Table 5 shows the highest resolution segment predicted under different models, including the ground truth, for different RTT values. We mark (using *) the correct decisions in each case. We see that ECON makes the right decision (as ground truth) in 3 of the 4 cases; for the RTT=150ms case, ECON underpredicts the throughput, resulting in a lower resolution segment being played (though the segment is transferred successfully in the segment duration time). By contrast, the best alternative models, He05-EWMA and Yin15-HM, correctly predict the resolution only in the first two cases, where RTT is not very high. Further, by using the actual *scwnd* value to estimate the amount of data that can be sent in the next segment duration, ECON's predictions are more accurate.

8 RELATED WORK

Existing TCP modeling work can be broadly categorized into (i) formula-based (FB) and (ii) history-based (HB) models. FB predictors rely on analytical models to characterize the TCP performance as a function of the underlying network [17, 20, 24, 55, 59, 63, 64]. HB predictors, on the other hand, employ time-series based techniques to forecast TCP performance based on historically observed values [34, 73, 77].

FB models: The PFTK model [63] characterizes the steady-state TCP (Reno) throughput as a function of loss probability (p) and RTT. Newer models correct errors in PFTK [20] 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 [17].

Several other stochastic TCP modeling approaches have been proposed but assume specific distributions for packet losses [5, 7, 59]. Our empirical measurements show that loss probability depends on cwnd, and does not seem to follow any predetermined distribution.

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 [31], but assume that the congestion points in the link are known. Hespanha et al. [36] 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) [34] are HB models that predict TCP throughput based on previously observed throughput values, using their namesake time-series prediction techniques (see §5.1). Last sample (Sun16-LS) [72] and Harmonic mean (Yin15-HM) [83] are HB models that were employed in recent works [43, 44, 72, 83] to select the optimal bitrate in HTTP-based adaptive video streaming.

Prior approaches have also employed learning algorithms, such as Support Vector Regression, for TCP performance modeling [12, 48, 58]. However, such approaches typically require additional features that are not always observable, such as queueing delay and available bandwidth [58], and may also require non-trivial parameter tuning [12].

Models for specific scenarios and variants: There are several works that focus specifically on short flows [16, 56] but also assume that losses are independent. For large flows, DualPats leverages the correlation between TCP throughput and flow size to predict performance [52]. There are also works that focus specifically on deriving models for parallel TCP connections [8, 53] and for wireless TCP connections [30, 45, 46, 62]. 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 [13, 50, 60]. 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 [10, 25, 51, 64, 68, 70, 80] 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 that have a specifiable cwnd evolution; we will investigate such extensions as part of future work.

A new TCP variant, BBR [15], is quickly gaining popularity. BBR's congestion control is not loss-based, and uses network probing to estimate cwnd. We are currently exploring how ECON can be extended to such non-loss–based algorithms.

HTTP models: Zarifis et al. recently proposed an approach to estimate HTTP/2 performance [84]. However, this approach relies on the availability of existing HTTP/1.1 traces. Heidemann et al. [35] 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 [18, 54], and not throughput modeling, which is the focus of our work.

Optimizing video performance: There have been recent works that focus on optimizing video performance. Oboe [6] leverages the piecewise stationarity of TCP throughput to automatically tune ABR configurations in real-time. By contrast, ECON chooses the optimal bitrate by predicting TCP throughput. Salsify [29] is a new architecture that integrates a video codec and a network transport protocol (UDP). We will look into how ECON can be integrated with Salsify as part of our future work.

9 CONCLUSION AND FUTURE WORK

ECON presents a scalable and systematic model to easily evaluate the performance of different network choices. 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 leveraging the empirically-augmented model, ECON removes the limiting assumptions made by current models. ECON's application level model for latency and HTTP is built on top of the TCP model.

Evaluation results across several networks show that our ECON TCP model predicts throughput and latency with an error of *6%–16%*; the ECON HTTP predictions have an error of less than 8%. We demonstrate ECON's applicability to improving network performance by (i) enabling a Web server application to accurately choose between HTTP/1.1 and HTTP/2 to reduce Web PLT, and (ii) helping a video server/client choose the optimal video bitrate to maximize video quality without rebuffering.

While the focus of this paper was on developing the ECON modeling approach, we do plan to implement our model-driven solutions for automatically selecting video bitrates and for switching between HTTP/1.1 and HTTP/2. We expect the system to be implemented primarily in the application; we envision the TCP model building and parameter acquisition to be performed at the OS, and the application level analysis to be done at the application layer.

ACKNOWLEDGMENT

This work was supported by NSF grants 1566260, 1717588, 1750109, and 1909356.

A APPENDIX: MODELING TCP CUBIC

Deriving $E[M]$: Given the expression for M and $Pr(M)$ from §3.3, we derive $E[M] = \sum_i i \cdot Pr(M = i)$, by conditioning over M and generalizing to c parallel connections:

$$E[M] = \sum_{n=1}^{\infty} \sum_{m=1}^{cwnd_n} \left(\left(\sum_{j=1}^{n-1} C((j-1) \cdot RTT - K)^3 + \frac{scwnd}{1-\beta} \right) + m \right) \times \left(\prod_{j=1}^{n-1} (1 - p(cwnd_j, c))^{cwnd_j} \right) (1 - p(cwnd_n, c))^{m-1} p(cwnd_n, c)$$

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

$$E[X] = \sum_{n=1}^{\infty} n(1 - (1 - p(cwnd_n, c))^{cwnd_n}) \times \prod_{j=1}^{n-1} (1 - p(cwnd_j, c))^{cwnd_j}$$

Deriving $E[N]$: Recall that $E[N]$ is the expected number of packets sent during the TDP. Given $E[M]$ and $E[X]$, we can derive $E[N]$ using $E[N] = (E[M] - 1) + cwnd_{E[X]}$ via Eq. (3).

Deriving B: Finally, the throughput can be derived, via Eq. (2), as $B = E[N]/((E[X] + 1) \cdot RTT)$.

REFERENCES

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

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

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

[4] Nginx. https://nginx.org/en/, 2017.

[5] 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.

[6] Akhtar, Z., Nam, Y. S., Govindan, R., Rao, S., Chen, J., Katz-Bassett, E., Ribeiro, B., Zhan, J., and Zhang, H. Oboe: Auto-tuning Video ABR Algorithms to Network Conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary, 2018), pp. 44–58.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] Cao, Y., Nejati, J., Wajahat, M., Balasubramanian, A., and Gandhi, A. Deconstructing the Energy Consumption of the Mobile Page Load. In *Proceedings of the 2017 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science* (Urbana-Champaign, IL, USA, 2016), SIGMETRICS '17.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] Cinlar, E. Markov Renewal Theory. *Advances in Applied Probability 1*, 2 (1969), 123–187.

[22] DASH: Dynamic Adaptive Streaming over HTTP. https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP.

[23] Dimopoulos, G., Leontiadis, I., Barlet-Ros, P., and Papagiannaki, K. Measuring Video QoE from Encrypted Traffic. In *Proceedings of the 2016 Internet Measurement Conference* (Santa Monica, CA, USA, 2016), pp. 513–526.

[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] Github repo for ECON data. https://github.com/DavyCao/ECON-Data.

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

[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] Fouladi, S., Emmons, J., Orbay, E., Wu, C., Wahby, R. S., and Winstein, K. Salsify: low-latency network video through tighter integration between a video codec and a transport protocol. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)* (2018), pp. 267–282.

[30] 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.

[31] 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.

[32] 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.

[33] HTTP Archive. http://httparchive.org/.

[34] 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.

[35] 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.

[36] 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.

[37] HTTP Live Streaming. https://en.wikipedia.org/wiki/HTTP_Live_Streaming.

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

[39] Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., and Watson, M. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, IL, USA, 2014), pp. 187–198.

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

[41] Jiang, H., Liu, Z., Wang, Y., Lee, K., and Rhee, I. Understanding bufferbloat in cellular networks. In *Proceedings of the 2012 ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design* (New York, NY, USA, 2012), CellNet '12, ACM, pp. 1–6.

[42] Jiang, J., Sekar, V., Milner, H., Shepherd, D., Stoica, I., and Zhang, H. CFA: A Practical Prediction System for Video QoE Optimization. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA, USA, 2016), pp. 137–150.

[43] Jiang, J., Sekar, V., and Sun, Y. Dda: Cross-session throughput prediction with applications to video bitrate selection. *arXiv preprint arXiv:1505.02056* (2015).

[44] Jiang, J., Sekar, V., and Zhang, H. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (TON) 22*, 1 (2014), 326–340.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] Li, H. Web page replay go. http://bit.ly/2p9fUe4.

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

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

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

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

[58] 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.

[59] 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.

[60] 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.

[61] Nazir, S., Hossain, Z., Secchi, R., Broadbent, M., Petlund, A., and Fairhurst, G. Performance evaluation of congestion window validation for dash transport. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop* (Singapore, Singapore, 2014), NOSSDAV '14, pp. 67:67–67:72.

[62] 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.

[63] 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.

[64] 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.

[65] Quic: a multiplexed stream transport over UDP. https://www.chromium.org/quic.

[66] 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.

[67] Ruamviboonsuk, V., Netravali, R., Uluyol, M., and Madhyastha, H. V. Vroom: Accelerating the mobile web with server-aided dependency resolution. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2017), SIGCOMM '17, ACM, pp. 390–403.

[68] 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.

[69] Sandvine. Global Internet Phenomena Report. https://www.sandvine.com/hubfs/downloads/archive/2014-2h-global-internet-phenomena-report.pdf, 2014.

[70] 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.

[71] Sivakumar, A., Puzhavakath Narayanan, S., Gopalakrishnan, V., Lee, S., Rao, S., and Sen, S. PARCEL: Proxy Assisted BRowsing in Cellular Networks for Energy and Latency Reduction. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies* (Sydney, Australia, 2014), CoNEXT '14, pp. 325–336.

[72] Sun, Y., Yin, X., Jiang, J., Sekar, V., Lin, F., Wang, N., Liu, T., and Sinopoli, B. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil, 2016), pp. 272–285.

[73] 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.

[74] T-Mobile. T-Mobile Network Performance Data. https://www.t-mobile.com/coverage/network-performance-data.

[75] Thang, T. C., Ho, Q., Kang, J. W., and Pham, A. T. Adaptive streaming of audiovisual content using mpeg dash. *IEEE Transactions on Consumer Electronics 58*, 1 (2012), 78–85.

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

[77] 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.

[78] 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.

[79] Verizon. 4G LTE speeds vs. your home network. https://www.verizonwireless.com/articles/4g-lte-speeds-vs-your-home-network/.

[80] 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.

[81] 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.

[82] WProfX: Significantly Improving Web Page Performance. wprofx.cs.stonybrook.edu.

[83] Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. A control-theoretic approach for dynamic adaptive video streaming over http. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 325–338.

[84] 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.

[85] Zou, X. K., Erman, J., Gopalakrishnan, V., Halepovic, E., Jana, R., Jin, X., Rexford, J., and Sinha, R. K. Can accurate predictions improve video streaming in cellular networks? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* (2015), ACM, pp. 57–62.