Pantheon: the training ground for Internet congestion-control research

Francis Y. Yan, Jestin Ma, and Greg D. Hill, ***Stanford University;***
Deepti Raghavan, ***Massachusetts Institute of Technology;***
Riad S. Wahby, Philip Levis, and Keith Winstein, ***Stanford University***

https://www.usenix.org/conference/atc18/presentation/yan-francis

This paper is included in the Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC '18).

July 11–13, 2018 • Boston, MA, USA

ISBN 978-1-939133-02-1

Open access to the Proceedings of the 2018 USENIX Annual Technical Conference is sponsored by USENIX.

Pantheon: the training ground for Internet congestion-control research

Francis Y. Yan[†], Jestin Ma[†], Greg D. Hill[†], Deepti Raghavan[¶], Riad S. Wahby[†], Philip Levis[†], Keith Winstein[†]

[†]Stanford University, [¶]Massachusetts Institute of Technology

Abstract

Internet transport algorithms are foundational to the performance of network applications. But a number of practical challenges make it difficult to evaluate new ideas and algorithms in a reproducible manner. We present the Pantheon, a system that addresses this by serving as a community "training ground" for research on Internet transport protocols and congestion control (https://pantheon.stanford.edu). It allows network researchers to benefit from and contribute to a common set of benchmark algorithms, a shared evaluation platform, and a public archive of results.

We present three results showing the Pantheon's value as a research tool. First, we describe a measurement study from more than a year of data, indicating that congestion-control schemes vary dramatically in their relative performance as a function of path dynamics. Second, the Pantheon generates calibrated network emulators that capture the diverse performance of real Internet paths. These enable reproducible and rapid experiments that closely approximate real-world results. Finally, we describe the Pantheon's contribution to developing new congestion-control schemes, two of which were published at USENIX NSDI 2018, as well as data-driven neural-network-based congestion-control schemes that can be trained to achieve good performance over the real Internet.

1 Introduction

Despite thirty years of research, Internet congestion control and the development of transport-layer protocols remain cornerstone problems in computer networking. Congestion control was originally motivated by the desire to avoid catastrophic network collapses [22], but today it is responsible for much more: allocating capacity among contending applications, minimizing delay and variability, and optimizing high-level metrics such as video rebuffering, Web page load time, the completion of batch jobs in a datacenter, or users' decisions to engage with a website.

In the past, the prevailing transport protocols and congestion-control schemes were developed by researchers [18, 22] and tested in academic networks or other small testbeds before broader deployment across the Internet. Today, however, the Internet is more diverse, and studies on academic networks are less likely to generalize to, e.g., CDN nodes streaming video at 80 Gbps [26], smartphones on overloaded mobile networks [8], or security cameras connected to home Wi-Fi networks.

As a result, operators of large-scale systems have begun to develop new transport algorithms in-house. Operators can deploy experimental algorithms on a small subset of their live traffic (still serving millions of users), incrementally improving performance and broadening deployment as it surpasses existing protocols on their live traffic [1, 7, 24]. These results, however, are rarely reproducible outside the operators of large services.

Outside of such operators, research is usually conducted on a much smaller scale, still may not be reproducible, and faces its own challenges. Researchers often create a new testbed each time—interesting or representative network paths to be experimented over—and must fight "bit rot" to acquire, compile, and execute prior algorithms in the literature so they can be fairly compared against. Even so, results may not generalize to the wider Internet. Examples of this pattern in the academic literature include Sprout [42], Verus [43], and PCC [12].

This paper describes the **Pantheon**: a distributed, collaborative system for researching and evaluating end-to-end networked systems, especially congestion-control schemes, transport protocols, and network emulators. The Pantheon has four parts:

1. a software library containing a growing collection of transport protocols and congestion-control algorithms, each verified to compile and run by a continuous-integration system, and each exposing the same interface to start or stop a full-throttle flow,

2. a diverse testbed of network nodes on wireless and wired networks around the world, including cellular networks in Stanford (U.S.), Guadalajara (Mexico), São Paulo (Brazil), Bogotá (Colombia), New Delhi (India), and Beijing (China), and wired networks in all of the above locations as well as London (U.K.), Iowa (U.S.), Tokyo (Japan), and Sydney (Australia),
3. a collection of network emulators, each calibrated to match the performance of a real network path between two nodes, or to capture some form of pathological network behavior, and
4. a continuous-testing system that regularly evaluates the Pantheon protocols over the real Internet between pairs of testbed nodes, across partly-wireless and all-wired network paths, and over each of the network emulators, in single- and multi-flow scenarios, and publicly archives the resulting packet traces and analyses at https://pantheon.stanford.edu.

The Pantheon's calibrated network emulators address a tension that protocol designers face between experimental realism and reproducibility. Simulators and emulators are reproducible and allow rapid experimentation, but may fail to capture important dynamics of real networks [15, 16, 31]. To resolve this tension, the Pantheon generates network emulators calibrated to match real Internet paths, graded by a novel figure of merit: their accuracy in matching the performance of a set of transport algorithms. Rather than focus on the presence or absence of modeled phenomena (jitter, packet loss, reordering), this metric describes how well the end-to-end performance (e.g., throughput, delay, and loss rate) of a set of algorithms, run over the emulated network, matches the corresponding performance statistics of the same algorithms run over a real network path.

Motivated by the success of ImageNet [11, 17] in the computer-vision community, we believe a common reference set of runnable benchmarks, continuous experimentation and improvement, and a public archive of results will enable faster innovation and more effective, reproducible research. Early adoption by independent research groups provides encouraging evidence that this is succeeding.

Summary of results:

- Examining more than a year of measurements from the Pantheon, we find that transport performance is highly variable across the type of network path, bottleneck network, and time. There is no single existing protocol that performs well in all settings. Furthermore, many protocols perform differently from how their creators intended and documented (§4).
- We find that a small number of network-emulator parameters (bottleneck link rate, isochronous or memoryless packet inter-arrival timing, bottleneck buffer size, stochastic per-packet loss rate, and propagation delay) is sufficient to replicate the performance of a diverse library of transport protocols (with each protocol matching its real-world throughput and delay to within 17% on average), in the presence of both natural and synthetic cross traffic. These results go against some strains of thought in computer networking, which have focused on building detailed network emulators (with mechanisms to model jitter, reordering, the arrival and departure of cross traffic, MAC dynamics, etc.), while leaving the questions open of how to configure an emulator to accurately model real networks and how to quantify the emulator's overall fidelity to a target (§5).
- We discuss three new approaches to congestion control that are using the Pantheon as a shared evaluation testbed, giving us encouragement that it will prove useful as a community resource. Two are from research groups distinct from the present authors, and were published at USENIX NSDI 2018: Copa [2] and Vivace [13]. We also describe our own data-driven designs for congestion control, based on neural networks that can be trained on a collection of the Pantheon's emulators and in turn achieve good performance over real Internet paths (§6).

2 Related work

Pantheon benefits from a decades-long body of related work in Internet measurement, network emulation, transport protocols, and congestion-control schemes.

Tools for Internet measurement. Systems like PlanetLab [10], Emulab [40], and ORBIT [30] provide measurement nodes for researchers to test transport protocols and other end-to-end applications. PlanetLab, which was in wide use from 2004–2012, at its peak included hundreds of nodes, largely on well-provisioned (wired) academic networks around the world. Emulab allows researchers to run experiments over configurable network emulators and on Wi-Fi links within an office building.

While these systems are focused on allowing researchers to borrow nodes and run their own tests, the Pantheon operates at a higher level of abstraction. Pantheon includes a *single* community software package that researchers can contribute algorithms to. Anybody can run any of the algorithms in this package, including over Emulab or any network path, but Pantheon also hosts a common repository of test results (including raw packet traces) of scripted comparative tests.

Network emulation. Congestion-control research has long used network simulators, e.g., ns-2 [28], as well as real-time emulators such as Dummynet [6, 33], NetEm [20], Mininet [19], and Mahimahi [27].

These emulators provide increasing numbers of parameters and mechanisms to recreate different network behaviors, such as traffic shapers, policers, queue disciplines, stochastic i.i.d. or autocorrelated loss, reordering, bit errors, and MAC dynamics. However, properly setting these parameters to emulate a particular target network remains an open problem.

One line of work has focused on improving emulator precision in terms of the level of detail and fidelity at modeling small-scale effects (e.g., "Two aspects influence the accuracy of an emulator: how detailed is the model of the system, and how closely the hardware and software can reproduce the timing computed by the model" [6]). Pantheon takes a different approach, instead focusing on accuracy in terms of how well an emulator recreates the performance of a set of transport algorithms.

Congestion control. Internet congestion control has a deep literature. The original DECBit [32] and Tahoe [22] algorithms responded to one-bit feedback from the network, increasing and decreasing a congestion window in response to acknowledgments and losses. More recently, researchers have tried to formalize the protocol-design process by generating a congestion-control scheme as a function of an objective function and prior beliefs about the network and workload. Remy [37, 41] and PCC [12] are different kinds of "learned" schemes [35]. Remy uses an offline optimizer that generates a decision tree to optimize a global utility score based on network simulations. PCC uses an online optimizer that adapts its sending rate to maximize a local utility score in response to packet losses and RTT samples. In our current work (§ 6), we ask whether it is possible to quickly train an algorithm from first principles to produce good *global* performance on real Internet paths.

3 Design and implementation

This section describes the design and implementation of the Pantheon, a system that automatically measures the performance of many transport protocols and congestion-control schemes across a diverse set of network paths. By allowing the community to repeatably evaluate transport algorithms in scripted comparative tests across real-world network paths, posted to a public archive of results, the Pantheon aims to help researchers develop and test algorithms more rapidly and reproducibly.

Below, we demonstrate several uses for Pantheon: comparing existing congestion-control schemes on real-world networks (§4); calibrating network emulators that accurately reproduce real-world performance (§5); and designing and testing new congestion-control schemes (§6).

Label	Scheme	LoC
Copa	Copa [2]	46
LEDBAT	LEDBAT/μTP [36] (`libutp`)	48
PCC	PCC† [12]	46
QUIC	QUIC Cubic [24] (`proto-quic`)	119
SCReAM	SCReAM [23]	541
Sprout	Sprout† [42]	46
Tao	RemyCC "100x" (2014) [37]	43
BBR	TCP BBR [7]	52
Cubic	TCP Cubic [18] (Linux default)	30
Vegas	TCP Vegas [5]	50
Verus	Verus† [43]	43
—	Vivace [13]	37
WebRTC	WebRTC media [4] in Chromium	283
—	FillP (work in progress)	41
Indigo	LSTM neural network (work in progress)	35

Figure 1: The Pantheon's transport schemes (§3.1.1) and the labels used for them in figures in this paper. Shown are the number of lines of Python, C++, or Javascript code in each wrapper that implements the common abstraction. Schemes marked † are modified to reduce MTU.

3.1 Design overview

Pantheon has three components: (1) a software repository containing pointers to transport-protocol implementations, each wrapped to expose a common testing interface based on the abstraction of a full-throttle flow; (2) testing infrastructure that runs transport protocols in scripted scenarios, instruments the network to log when each packet was sent and received, and allows flows to be initiated by nodes behind a network address translator (NAT); and (3) a global observatory of network nodes, enabling measurements across a wide variety of paths. We describe each in turn.

3.1.1 A collection of transport algorithms, each exposing the same interface

To test each transport protocol or congestion-control scheme on equal footing, Pantheon requires it to expose a common abstraction for testing: a full-throttle flow that runs until a sender process is killed. The simplicity of this interface has allowed us (and a few external contributors so far) to write simple wrappers for a variety of schemes and contribute them to the Pantheon, but limits the kinds of evaluations the system can do.[1]

Figure 1 lists the currently supported schemes, plus the size (in lines of code) of a wrapper script to expose the required abstraction. For all but three schemes, no modification was required to the existing implementation. The remaining three had a hard-coded MTU size and

[1] For example, the interface allows measurements of combinations of long-running flows (with timed events to start and stop a flow), but does not allow the caller to run a scheme until it has transferred exactly x bytes. This means that the Pantheon cannot reliably measure the flow-completion time of a mix of small file transfers.

required a small patch to adjust it for compatibility with our network instrumentation; please see §3.1.2 below.

As an example, we describe the Pantheon's wrapper to make WebRTC expose the interface of a full-throttle flow. The Pantheon tests the Chromium implementation of WebRTC media transfer [4] to retrieve and play a video file. The wrapper starts a Chromium process for the sender and receiver, inside a virtual X frame buffer, and provides a signaling server to mediate the initial connection. This comprises about 200 lines of JavaScript.

Pantheon is designed to be easily extended; researchers can add a new scheme by submitting a pull request that adds a submodule reference to their implementation and the necessary wrapper script. Pantheon uses a continuous-integration system to verify that each proposed scheme builds and runs in emulation.

3.1.2 Instrumenting network paths

For each IP datagram sent by the scheme, Pantheon's instrumentation tracks the size, time sent, and (if applicable) time received. Pantheon allows either side (sender or receiver) to initiate the connection, even if one of them is behind a NAT, and prevents schemes from communicating with nodes other than the sender and receiver. To achieve this, Pantheon creates a virtual private network (VPN) between the endpoints, called a *Pantheon-tunnel*, and runs all traffic over this VPN.

Pantheon-tunnel comprises software controlling a virtual network device (TUN) [39] at each endpoint. The software captures all IP datagrams sent to the local TUN, assigns each a unique identifier (UID), and logs the UID and a timestamp. It then encapsulates the packet and its UID in a UDP datagram, which it transmits to the other endpoint via the path under test. The receiving endpoint decapsulates, records the UID and arrival time, and delivers the packet to its own Pantheon-tunnel TUN device.

This arrangement has two main advantages. First, UIDs make it possible to unambiguously log information about every packet (e.g., even if packets are retransmitted or contain identical payloads). Second, either network endpoint can be the sender or receiver of an instrumented network flow over an established Pantheon-tunnel, even if it is behind a NAT (as long as one endpoint has a routable IP address to establish the tunnel).

Pantheon-tunnel also has disadvantages. First, encapsulation costs 36 bytes (for the UID and headers), reducing the MTU of the virtual interface compared to the path under test; for schemes that assume a fixed MTU, Pantheon patches the scheme accordingly. Second, because each endpoint records a timestamp to measure the send and receive time of each datagram, accurate timing requires the endpoints' clocks to be synchronized; endpoints use NTP [29] for this purpose. Finally, Pantheon-tunnel makes all traffic appear to the network as UDP, meaning it cannot measure the effect of a network's discrimination based on the IP protocol type.[2]

Evaluation of Pantheon-tunnel. To verify that Pantheon-tunnel does not substantially alter the performance of transport protocols, we ran a calibration experiment to measure the tunnel's effect on the performance of three TCP schemes (Cubic, Vegas, and BBR). We ran each scheme 50 times inside and outside the tunnel for 30 seconds each time, between a colocation facility in India and the EC2 India datacenter, measuring the mean throughput and 95th-percentile per-packet one-way delay of each run.[3] We ran a two-sample Kolmogorov-Smirnov test for each pair of statistics (the 50 runs inside vs. outside the tunnel for each scheme's throughput and delay). No test found a statistically significant difference below $p < 0.2$.

3.1.3 A testbed of nodes on interesting networks

We deployed observation nodes in countries around the world, including cellular (LTE/UMTS) networks in Stanford (USA), Guadalajara (Mexico), São Paulo (Brazil), Bogotá (Colombia), New Delhi (India), and Beijing (China), wired networks in all of the above locations as well as London (U.K.), Iowa (U.S.), Tokyo (Japan), and Sydney (Australia), and a Wi-Fi mesh network in Nepal. These nodes were provided by a commercial colocation facility (Mexico, Brazil, Colombia, India), by volunteers (China and Nepal), or by Google Compute Engine (U.K., U.S., Tokyo, Sydney).

We found that hiring a commercial colocation operator to maintain LTE service in far-flung locations has been an economical and practical approach; the company maintains, debugs, and "tops up" local cellular service in each location in a way that would otherwise be impractical for a university research group. However, this approach limits us to available colocation sites and ones where we receive volunteered nodes. We are currently bringing up a volunteered node with cellular connectivity in Saudi Arabia and welcome further contributions.

3.2 Operation and testing methods

The Pantheon frequently benchmarks its stable of congestion-control schemes over each path to create an archive of real-world network observations. On each path, Pantheon runs multiple benchmarks per week. Each benchmark follows a software-defined scripted workload (e.g., a single flow for 30 seconds; or multiple flows of

[2] Large-scale measurements by Google [24] have found such discrimination, after deployment of the QUIC UDP protocol, to be rare.

[3] For BBR running outside the tunnel, we were only able to measure the average throughput (not delay). Run natively, BBR's performance relies on TCP segmentation offloading [9], which prevents a precise measurement of per-packet delay without the tunnel's encapsulation.

cross traffic, arriving and departing at staggered times), and for each benchmark, Pantheon chooses a random ordering of congestion-control schemes, then tests each scheme in round-robin fashion, repeating until every scheme has been tested 10 times (or 3 for partly-cellular paths). This approach mirrors the evaluation methods of prior academic work ([12, 42, 43]).

During an experiment, both sides of a path repeatedly measure their clock offset to a common NTP server and use these to calculate a corrected one-way delay of each packet. After running an experiment, a node calculates summary statistics (e.g., mean throughput, loss rate, and 95th-percentile one-way delay for each scheme) and uploads its logs (packet traces, analyses, and plots) to AWS S3 and the Pantheon website (https://pantheon.stanford.edu).

4 Findings

The Pantheon has collected and published measurements of a dozen protocols taken over the course of more than a year. In this section, we give a high-level overview of some key findings in this data, focusing on the implications for research and experimental methodology. We examine comparative performance between protocols rather than the detailed behavior of particular protocols, because comparative analyses provide insight into which protocol end hosts should run in a particular setting.

To ground our findings in examples from concrete data, we select one particular path: AWS Brazil to Colombia. This path represents the performance a device in Colombia would see downloading data from properly geo-replicated applications running in AWS (Brazil is the closest site).

Finding 1: Which protocol performs best varies by path. Figure 2a shows the throughput and delay of 12 transport protocols from AWS Brazil to a server in Colombia, with an LTE modem from a local carrier (Claro).[4] Figure 2b shows the throughput and delay for the same protocols from a node at Stanford University with a T-Mobile LTE modem, to a node in AWS California. The observed performance varies significantly. In Brazil-Colombia, PCC is within 80% of the best observed throughput (QUIC) but with delay 20 times higher than the lowest (SCReAM). In contrast, for Stanford-California, PCC has only 52% of the best observed throughput (Cubic) and the lowest delay. The Sprout scheme, developed by one of the present authors, was designed for cellular networks in the U.S. and performs well in that setting (Figure 2b), but poorly on other paths.

[4] All results in this paper and supporting raw data can be found in the Pantheon archive; e.g. the experiment indicated as P123 can be found at https://pantheon.stanford.edu/result/123/.

These differences are not only due to long haul paths or geographic distance. Figure 2c shows the performance of the transport protocols from AWS Brazil to a wired device in Colombia. Performance is completely different. Delays, rather than varying by orders of magnitude, differ by at most 32%. At the same time, some protocols are strictly better: QUIC (Cubic) and (TCP) Cubic have both higher throughput and lower delay than BBR and Verus.

Differences are not limited to paths with cellular links. Figure 2e shows performance between Stanford and AWS California using high-bandwidth wired links and Figure 2f shows performance between the Google Tokyo and Sydney datacenters. While in both cases PCC shows high throughput and delay, in the AWS case BBR is better in throughput while between Google data centers it provides 34% less throughput. Furthermore, LEDBAT performs reasonably well on AWS, but has extremely low throughput between Google datacenters.

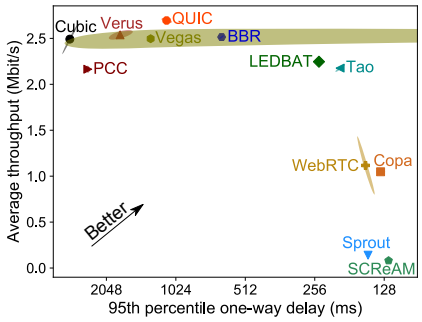
This suggests that evaluating performance on a small selection (or, in the worst case, just one) of paths can lead to misleadingly positive results, because they are not generalizable to a wide range of paths.

Finding 2: Which protocol performs best varies by path direction. Figure 2d shows the performance of the opposite direction of the path, from the same device with cellular connection in Colombia to AWS Brazil. This configuration captures the observed performance of uploading a photo or streaming video through a relay.

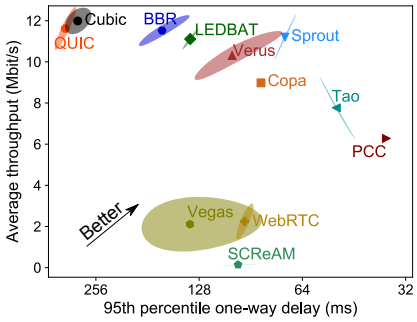
In the Brazil to Colombia direction, QUIC strictly dominates Vegas, providing both higher throughput and lower delay. In the opposite direction, however, the tradeoff is less clear: Vegas provides slightly lower throughput with a significant (factor of 9) decrease in delay. Similarly, in the Brazil to Colombia direction, WebRTC provides about half the throughput of LEDBAT while also halving delay; in the Colombia to Brazil direction, WebRTC is strictly worse, providing one third the throughput while quadrupling delay.

This indicates that evaluations of network transport protocols need to explicitly measure both directions of a path. On the plus side, a single path can provide two different sets of conditions when considering whether results generalize.

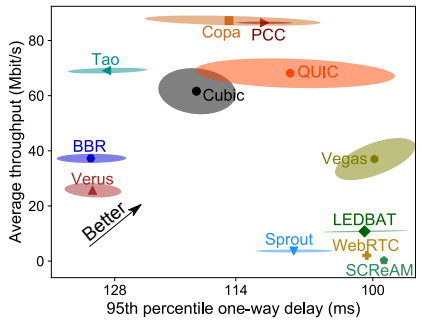
Finding 3: Protocol performance varies in time and only slightly based on competing flows. Figure 2g shows the Brazil-Colombia path measured twice, separated by two days (the first measurement shown in open dots is the same as in Figure 2a). Most protocols see a strict degradation of performance in the second measurement, exhibiting lower throughput and higher delay. Cubic and PCC, once clearly distinguishable, merge to have equivalent performance. More interestingly, the performance of Vegas has 23% lower throughput, but cuts delay by more than a factor of 2.



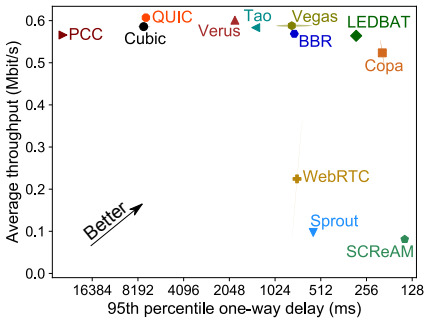
(a) AWS Brazil to Colombia (cellular), 1 flow, 3 trials. P1392.



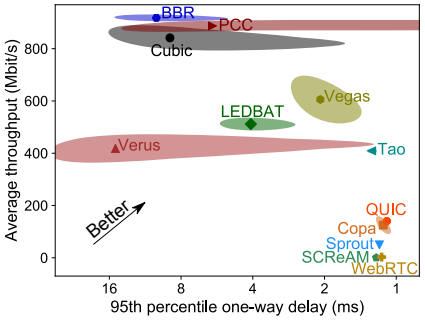
(b) Stanford to AWS California (cellular), 1 flow, 3 trials. P950.



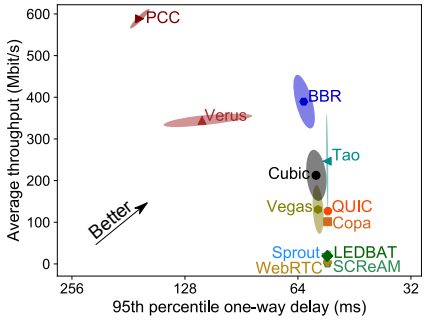
(c) AWS Brazil to Colombia (wired), 1 flow, 10 trials. P1271.



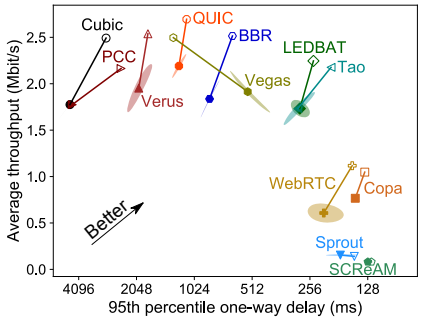
(d) Colombia to AWS Brazil (cellular), 1 flow, 3 trials. P1391.



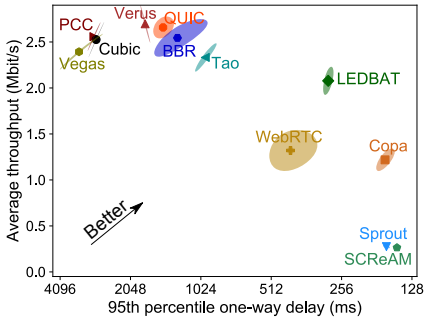
(e) Stanford to AWS California (wired), 3 flows, 10 trials. P1238.



(f) GCE Tokyo to GCE Sydney (wired), 3 flows, 10 trials. P1442.



(g) AWS Brazil to Colombia (cellular), 1 flow, 3 trials. 2 days after Figure 2a (shown in open dots). P1473.



(h) AWS Brazil to Colombia (cellular), 3 flows, 3 trials. P1405.

Figure 2: Compared with Figure 2a, scheme performance varies across the type of network path (Figure 2c), number of flows (Figure 2h), time (Figure 2g), data flow direction (Figure 2d), and location (Figure 2b). Figure 2e and 2f show that the variation is not limited to just cellular paths. The shaded ellipse around a scheme's dot represents the 1-σ variation across runs. Given a measurement ID, e.g. P123, the full result can be found at https://pantheon.stanford.edu/result/123/.

Finally, Figure 2h shows performance on the Brazil-Colombia path when 3 flows compete. Unlike in Figure 2a, PCC and Cubic dominate Vegas, and many protocols see similar throughput but at greatly increased latency (perhaps due to larger queue occupancy along the path).

This indicates that evaluations of network transport protocols need to not only measure a variety of paths, but also spread those measurements out in time. Furthermore, if one protocol is measured again, all of them need to be measured again for a fair comparison, as conditions may have changed. Cross traffic (competing flows) is an important consideration, but empirically has only a modest effect on relative performance. We do find that schemes that diverge significantly from traditional congestion control (e.g., PCC) exhibit poor fairness in some settings; in a set of experiments between Tokyo and Sydney (P1442), we observed the throughput ratios of three PCC flows to be 32:4:1. This seems to contradict fairness findings in the PCC paper and emphasizes the need for a shared evaluation platform across diverse paths.

5 Calibrated emulators

The results in Section 4 show that transport performance varies significantly over many characteristics, including time. This produces a challenge for protocol development and the ability of researchers to reproduce each others' results. One time-honored way to achieve controlled, reproducible results, at the cost of some realism, is to measure protocols in simulation or emulation [14] instead of the wild Internet, using tools like Dummynet [6, 33], NetEm [20], Mininet [19], or Mahimahi [27].

These tools each provide a number of parameters and mechanisms to recreate different network behaviors, and there is a traditional view in computer networking that the more fine-grained and detailed an emulator, the better. The choice of parameter values to faithfully emulate a particular target network remains an open problem.

In this paper, we propose a new figure of merit for network emulators: the degree to which an emulator can be substituted for the real network path in a full system, including the endpoint algorithm, without altering the system's overall performance. In particular, we define the emulator's accuracy as the average difference of the throughput and of the delay of a set of transport algorithms run over the emulator, compared with the same statistics from the real network path that is the emulator's target. The broader and more diverse the set of transport algorithms, the better characterized the emulator's accuracy will be: each new algorithm serves as a novel probe that could put the network into an edge case or unusual state that exercises the emulator and finds a mismatch.

In contrast to some conventional wisdom, we do not think that more-detailed network models are necessarily preferable. Our view is that this is an empirical question, and that more highly-parameterized network models create a risk of overfitting—but may be justified if lower-parameter models cannot achieve sufficient accuracy.

5.1 Emulator characteristics

We found that a five-parameter network model is sufficient to produce emulators that approximate a diverse variety of real paths, matching the throughput and delay of a range of algorithms to within 17% on average. The resulting calibrated emulators allow researchers to test experimental new schemes—thousands of parallel variants if necessary—in emulated environments that stand a good chance of predicting future real-world behavior.[5]

The five parameters are:

1. a bottleneck link rate,
2. a constant propagation delay,
3. a DropTail threshold for the sender's queue,
4. a stochastic loss rate (per-packet, i.i.d.), and
5. a bit that selects whether the link runs isochronously (all interarrival times equal), or with packet deliveries governed by a memoryless Poisson point process, characteristic of the observed behavior of some LTE networks [42].

To build emulators using these parameters, the Pantheon uses Mahimahi container-based network emulators [27]. In brief: Mahimahi gives the sender and receiver each its own isolated Linux network namespace, or container, on one host. An emulator is defined by a chain of nested elements, each one modeling a specific network effect: e.g., an `mm-loss` container randomly drops packets in the outgoing or incoming direction at a specified rate.

5.2 Automatically calibrating emulators to match a network path

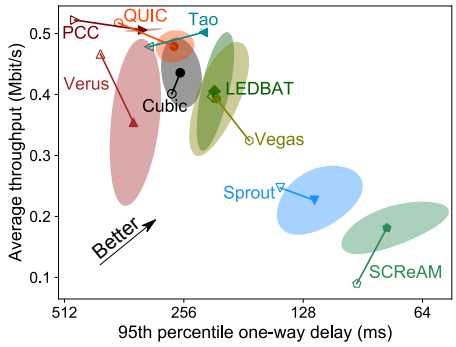
Given a set of results over a particular network path, Pantheon can generate an emulator that replicates the same results in about two hours, using an automated parameter-search process that we now describe.

To find an appropriate combination of emulator parameters, Pantheon searches the space using a non-linear optimization process that aims to find the optimal value for a vector x, which represents the <*rate*, *propagation delay*, *queue size*, *loss rate*> for the emulator.[6]

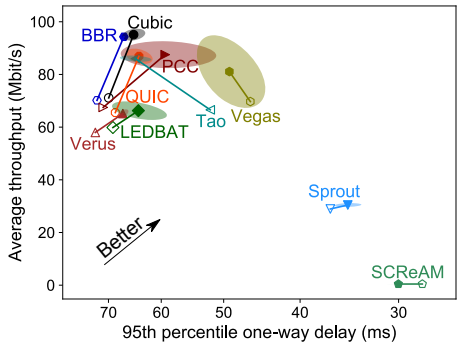
The optimization derives a replication error for each set of emulator parameters, $f(x)$, which is defined as the

[5] In a leave-one-out cross-validation experiment, we confirmed that emulators trained to match the performance of $n-1$ transport algorithms accurately predicted the unseen scheme's performance within about 20% (results not shown).

[6] The optimization is run twice, to choose between a constant rate or a Poisson delivery process.



(a) Nepal to AWS India (wireless), 1 flow, 10 trials. Mean replication error: 19.1%. P188.



(b) AWS California to Mexico (wired), 3 flows, 10 trials. Mean replication error: 14.4%. P1237.

Figure 3: Examples of per-scheme calibrated emulator errors. The filled dots represent real results over each network path; the open dots represent the corresponding result over the emulator that best replicates all of the results. Emulators for all-wired paths give better fidelity than emulators for partly-wireless paths (§5.3).

Path	Error (%)
Nepal to AWS India (Wi-Fi, 1 flow, P188)	19.1
AWS Brazil to Colombia (cellular, 1 flow, P339)	13.0
Mexico to AWS California (cellular, 1 flow, P196)	25.1
AWS Korea to China (wired, 1 flow, P361)	17.7
India to AWS India (wired, 1 flow, P251)	15.6
AWS California to Mexico (wired, 1 flow, P353)	12.7
AWS California to Mexico (wired, 3 flows, P1237)	14.4

Figure 4: Replication error of calibrated emulators on six paths with a single flow, and one path with three flows of staggered cross traffic.

Path	Feature change	Error (%)
China wired	link rate only	211.8
	add delay	211.8 → 189.7
	add buffer size	189.7 → 32.3
	add stochastic loss	32.3 → 17.7
Colombia cellular	constant → Poisson	23.7 → 13.0

Figure 5: Each of the emulator's five parameters is helpful in reducing replication error. For the China wired path, we started with a single parameter and added the other three features one by one, in the order of their contribution. The Colombia cellular path required jitter (Poisson deliveries) to achieve good accuracy.

average of the percentage changes between the real and emulated mean throughput, and the real and emulated mean 95th-percentile delay, across each of the set of reference transport algorithms. To minimize $f(x)$, nonlinear optimization is necessary because neither the mathematical expression nor the derivative of $f(x)$ is known. In addition, for both emulated and real world network paths, $f(x)$ is non-deterministic and noisy.

The Pantheon uses Bayesian optimization [25], a standard method designed for optimizing the output of a noisy function when observations are expensive to obtain and derivatives are not available.[7] The method starts with the assumption that the objective function, $f(x)$, is drawn from a broad prior (Gaussian is a standard choice and the one we use). Each sample (i.e., calculation of the emulator replication error for a given set of emulator parameters x) updates the posterior distribution for $f(x)$. Bayesian optimization uses an *acquisition function* to guide the algorithm's search of the input space to the next value x. We use the Spearmint [38] Bayesian-optimization toolkit, which uses "expected improvement" as its acquisition function. This function aims to maximize the expected improvement over the current best value [25].

5.3 Emulation results

We trained emulators that model six of Pantheon's paths, each for about 2 hours on 30 EC2 machines with 4 vCPUs each. Figure 3 shows per-scheme calibration results for two representative network paths, a wireless device in Nepal and a wired device in Mexico. Filled dots represent the measured *mean* performance of the scheme on the real network path, while the open dot represents the performance on the corresponding calibrated emulator. A closer dot means the emulator is better at replicating that scheme's performance.

We observe that the emulators roughly preserve the relative order of the mean performance of the schemes on each path. Figure 4 shows mean error in replicating the

[7] Each evaluation of $f(x)$ involves running all of Pantheon's congestion-control schemes in a scripted 30-second scenario, three times, across the emulated path. This is done in parallel, so each evaluation of $f(x)$ takes about 30 seconds of wall-clock time.

throughput and delay performance of all of Pantheon's congestion-control schemes by a series of emulators. To ensure each parameter is necessary, we measured the benefits of adding delay, queue size, and loss information to a base emulator that uses a constant rate, in replicating the China wired device path. For the cellular device path we measured the benefit of using a Poisson based link rate rather than a constant rate. As shown in Figure 5, each added parameter improves the emulator's fidelity.

Pantheon includes several calibrated emulators, and regularly runs the transport algorithms in single- and multi-flow scenarios over each of the emulators and publishes the results in its public archive. Researchers are also able to run the calibrated emulators locally.

In addition, Pantheon includes, and regularly evaluates schemes over, a set of "pathological" emulators suggested by colleagues at Google. These model extreme network behaviors seen in the deployment of the BBR scheme: very small buffer sizes, severe ACK aggregation, and token-bucket policers.

Overall, our intention is that Pantheon will contain a sufficient library of well-understood network emulators so that researchers can make appreciable progress evaluating schemes (perhaps thousands of variants at once) in emulation—with some hope that there will be fewer surprises when a scheme is evaluated over the real Internet.

6 Pantheon use cases

We envision Pantheon as a common evaluation platform and an aid to the development of new transport protocols and congestion-control schemes. In this section, we describe three different ways that Pantheon has been helpful. Two are based on experiences that other research groups have had using Pantheon to assist their efforts. The third is an example of a radical, data-driven congestion-control design based on neural networks learned directly from Pantheon's network emulators.

Case 1. Vivace: validating a new scheme in the real world. Dong et al. [13] describe a new congestion-control scheme called Vivace, the successor to PCC [12]. They contributed three variants of the scheme to Pantheon in order to evaluate and tune Vivace's performance, by examining Pantheon's packet traces and analyses of Vivace in comparison with other schemes across an array of real-world paths. This is consistent with Pantheon's goal of being a resource for the research community (§1).

Case 2. Copa: iterative design with measurements. Arun and Balakrishnan [2] describe another new scheme, Copa, which optimizes an objective function via congestion window and sending rate adjustments. In contrast to Vivace, which was deployed on Pantheon largely as a completed design, Copa used Pantheon as an integral part of the design process: the authors deployed a series of six prototypes, using Pantheon's measurements to inform each iteration. This demonstrates another use of Pantheon, automatically deploying and testing prototypes on the real Internet, and gathering *in vivo* performance data.

Case 3. Indigo: extracting an algorithm from data. As an extreme example of data-driven design, we present Indigo, a machine-learned congestion-control scheme whose design we extract from data gathered by Pantheon.

Using machine learning to train a congestion-control scheme for the real world is challenging. The main reason is that it is impractical to learn directly from the Internet: machine-learning algorithms often require thousands of iterations and hours to weeks of training time, meaning that paths evolve in time (§4) more quickly than the learning algorithm can converge. Pantheon's calibrated emulators (§5) provide an alternative: they are reproducible, can be instantiated many times in parallel, and are designed to replicate the behavior of congestion-control schemes. Thus, our high-level strategy is to train Indigo using emulators, then evaluate it in the real world using Pantheon.

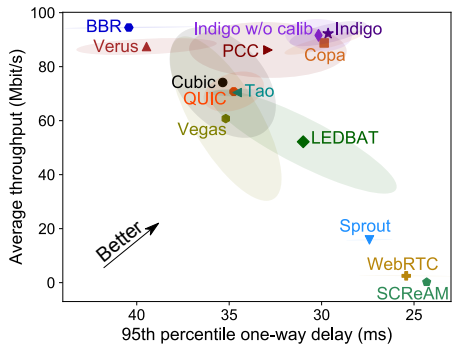
Indigo is one example of what we believe to be a novel family of data-driven algorithms enabled by Pantheon. Specifically, Pantheon facilitates realistic offline training and testing by providing a communal benchmark, evolving dataset, and calibrated emulators to allow approximately realistic offline training and testing. Below, we briefly describe Indigo's design; we leave a more detailed description to future work.

Overview of Indigo

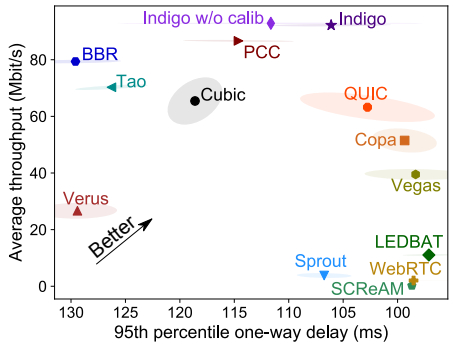
At its core, Indigo does two things: it observes the network state, and it adjusts its *congestion window*, i.e., the allowable number of in-flight packets. Observations occur each time an ACK is received, and their effect is to update Indigo's internal *state*, defined below. Indigo adjusts its congestion window every 10 ms. The state vector is:

1. An exponentially-weighted moving average (EWMA) of the queuing delay, measured as the difference between the current RTT and the minimum RTT observed during the current connection.
2. An EWMA of the sending rate, defined as the number of bytes sent since the last ACK'ed packet was sent, divided by the RTT.
3. An EWMA of the receiving rate, defined as the number of bytes received since the ACK preceding the transmission of the most recently ACK'ed packet, divided by the corresponding duration (similar to and inspired by TCP BBR's delivery rate [7]).
4. The current congestion window size.
5. The previous action taken.

Indigo stores the mapping from states to actions in a Long Short-Term Memory (LSTM) recurrent neural

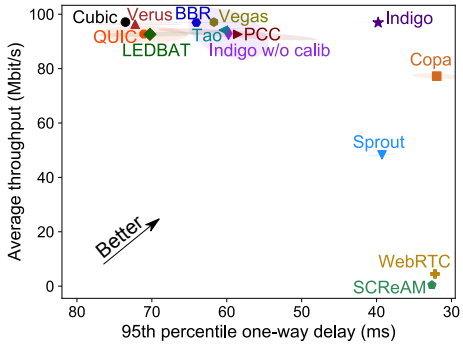


(a) Mexico to AWS California, 10 trials. P1272.

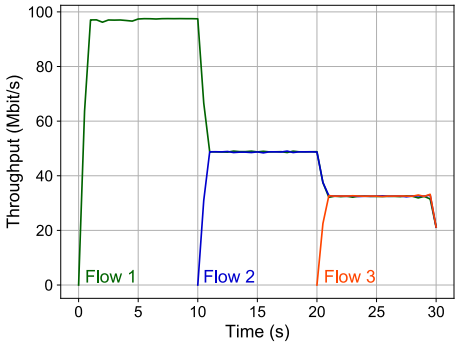


(b) AWS Brazil to Colombia, 10 trials. P1439.

Figure 6: Real wired paths, single flow. Indigo's performance is at the throughput/delay tradeoff frontier. Indigo without calibrated emulators ("Indigo w/o calib") gives worse performance.



(a) India to AWS India, 10 trials. P1476.



(b) Time-domain three-flow test. One trial in Figure 7a.

Figure 7: Real wired paths, multiple flows. Figure 7a shows the performance of all congestion-control schemes on multi-flow case. Figure 7b shows throughput vs. time for a three-flow run in Figure 7a starting 10 seconds apart. Indigo shares the bandwidth fairly.

network [21] with 1 layer of 32 hidden units (values chosen after extensive evaluation on the Pantheon). Indigo requires a *training phase* (described below) in which, roughly speaking, it learns a mapping from states to actions. Once trained and deployed, this mapping is fixed.

We note that there may be better parameter choices: number of hidden units, action space, state contents, etc. We have found that the above choices already achieve good performance; further improvements are future work. As one step toward validating our choices, we trained and tested several versions of Indigo with a range of hidden units, from 1 to 256, on an emulated network; choices between 16 and 128 yielded good performance.

Indigo's training phase. Indigo uses imitation learning [3, 34] to train its neural network. At a high level, this happens in two steps: first, we generate one or more *congestion-control oracles*, idealized algorithms that perfectly map states to correct actions, corresponding to links on which Indigo is to be trained. Then we apply a standard imitation learning algorithm that use these oracles to generate training data.

Of course, no oracle exists for real-world paths. Instead, we generate oracles corresponding to *emulated* paths; this is possible because Pantheon's emulators (§5) have few parameters. By the definition of an oracle, if we know the ideal congestion window for a given link, we have the oracle for the link: for any state, output whichever action results in a congestion window closest to the ideal value.

A key insight is that for emulated links, we can very closely approximate the ideal congestion window. For simple links with a fixed bandwidth and minimum one-way delay, the ideal window is given by the link's bandwidth-delay product (BDP) per flow. For calibrated emulators (which have DropTail queues, losses, etc.), we compute the BDP and then search near this value in emulation to find the best fixed congestion window size.

After generating congestion-control oracles corresponding to each training link, we use a state-of-the-art imitation learning algorithm called DAgger [34] to train the neural network. For each training link, DAgger trains Indigo's neural network as follows: first, it allows the

neural network to make a sequence of congestion-control decisions on the training link's emulator, recording the state vector that led to each decision. Next, it uses the congestion-control oracle to label the correct action corresponding to each recorded state vector. Finally, it updates the neural network by using the resulting state-action mapping as training data. This process is repeated until further training does not change the neural network.

Indigo's performance. In this section, we compare Indigo's performance with that of other congestion-control schemes, and we evaluate the effect of Pantheon's calibrated emulators on performance, versus only training on fixed-bandwidth, fixed-delay emulators.

We trained Indigo on 24 synthetic emulators uncorrelated to Pantheon's real network paths, and on the calibrated emulators (§5). The synthetic emulators comprise all combinations of (5, 10, 20, 50, 100, and 200 Mbps) link rate and (10, 20, 40, 80 ms) minimum one-way delay, with infinite buffers and no loss.

Indigo on Pantheon. We find that Indigo consistently achieves good performance. Figure 6 compares Indigo to other schemes in single flow on two wired paths. In both cases, Indigo is at the throughput/delay tradeoff frontier.

Figure 7 shows Indigo's performance in the multi-flow case. Figure 7a shows the performance of all of Pantheon's congestion-control schemes on a wired path from India to AWS India; Indigo is once again on the throughput/delay tradeoff frontier. Figure 7b is a time-domain plot of one trial from Figure 7a, suggesting that Indigo shares fairly, at least in some cases.

Benefit of calibrated emulators. Figures 6 and 7 also depict a variant of Indigo, "Indigo w/o calib," that is only trained on the synthetic emulators, but not the calibrated emulators. The version trained on calibrated emulators is always as least as good or better.

7 Discussion, limitations, and future work

Improving Pantheon. Pantheon would be more useful if it collected more data about congestion-control schemes. For instance, Pantheon currently gathers data only from a handful of nodes—vastly smaller than the evaluations large-scale operators can perform on even a small fraction of a billion-user population.

Moreover, geographic locality does not guarantee network path similarity: two nodes in the same city can have dramatically different network connections. Pantheon also only tests congestion-control schemes at full throttle; other traffic patterns (e.g., Web-like workloads) may provide researchers with valuable information (e.g., how their scheme affects page-load times).

Finally, Pantheon currently measures the interaction between multiple flows of cross-traffic governed by the same scheme, but we are working to make it measure interactions between different schemes. These measurements will help evaluate fairness in the real world.

Improving the calibrated emulators. Our current emulators replicate throughput and delay metrics only within 17% accuracy on average. An open question is whether we can improve emulator fidelity—especially on cellular paths—without risk of overfitting. Considering metrics other than 95th-percentile delay and mean throughput may be one path forward. Adding more schemes to Pantheon could also help—or it might reveal that the current set of emulator parameters, which we have empirically determined, is insufficient for some schemes.

Indigo. We have presented a case study on Indigo, a data-driven approach to congestion-control design that crucially relies on Pantheon's family of emulators. Indigo's trained model is complex and may have unknown failure modes, but the results to date demonstrate how Pantheon can enable new approaches to protocol design.

8 Conclusion

The Pantheon is a collection of transport protocols and a distributed system of measurement points and network emulators for evaluating and developing them. By measuring many transport protocols and congestion-control algorithms across a diverse set of paths, Pantheon provides a training ground for studying and improving their performance. Furthermore, by generating calibrated emulators that match real-world paths, Pantheon enables researchers to measure protocols reproducibly and accurately.

Pantheon has assisted in the development of two recently-published congestion-control algorithms [2, 13], and has supported our own data-driven approach to protocol design. In other areas of computer science, community benchmarks and recurrent bakeoffs have fueled advances and motivated researchers to build on each others' work: ImageNet in computer vision, the TPC family and Sort Benchmarks for data processing, Kaggle competitions in machine learning, etc. We are hopeful that Pantheon will, over time, serve a similar role in computer networking.

Acknowledgments

We thank the USENIX ATC reviewers for their helpful comments and suggestions. We are grateful to Yuchung Cheng, Janardhan Iyengar, Michael Schapira, and Hari Balakrishnan for feedback throughout this project. This work was supported by NSF grant CNS-1528197, DARPA grant HR0011-15-2-0047, Intel/NSF grant CPS-Security-1505728, the Secure Internet of Things Project, and by Huawei, VMware, Google, Dropbox, Facebook, and the Stanford Platform Lab.

References

[1] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center TCP (DCTCP). In *ACM SIGCOMM computer communication review* (2010), vol. 40, ACM, pp. 63–74.

[2] ARUN, V., AND BALAKRISHNAN, H. Copa: Practical delay-based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2018), USENIX Association.

[3] BENGIO, S., VINYALS, O., JAITLY, N., AND SHAZEER, N. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR abs/1506.03099* (2015).

[4] BERGKVIST, A., BURNETT, D. C., JENNINGS, C., NARAYANAN, A., AND ABOBA, B. Webrtc 1.0: Real-time communication between browsers. *Working draft, W3C 91* (2012).

[5] BRAKMO, L. S., O'MALLEY, S. W., AND PETERSON, L. L. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM* (1994).

[6] CARBONE, M., AND RIZZO, L. Dummynet revisited. *SIGCOMM Comput. Commun. Rev. 40*, 2 (Apr. 2010), 12–20.

[7] CARDWELL, N., CHENG, Y., GUNN, C. S., YEGANEH, S. H., AND JACOBSON, V. BBR: Congestion-based congestion control. *Queue 14*, 5 (2016), 50.

[8] CHEN, J., SUBRAMANIAN, L., IYENGAR, J., AND FORD, B. TAQ: Enhancing fairness and performance predictability in small packet regimes. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 7:1–7:14.

[9] CHENG, Y., AND CARDWELL, N. Making Linux TCP fast.

[10] CHUN, B., CULLER, D., ROSCOE, T., BAVIER, A., PETERSON, L., WAWRZONIAK, M., AND BOWMAN, M. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review 33*, 3 (July 2003), 00–00.

[11] DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., AND FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 248–255.

[12] DONG, M., LI, Q., ZARCHY, D., GODFREY, P. B., AND SCHAPIRA, M. PCC: Re-architecting congestion control for consistent high performance. In *Presented as part of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2015).

[13] DONG, M., MENG, T., ZARCHY, D., ARSLAN, E., GILAD, Y., GODFREY, B., AND SCHAPIRA, M. PCC Vivace: Online-learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2018), USENIX Association.

[14] FALL, K., AND FLOYD, S. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *SIGCOMM Comput. Commun. Rev. 26*, 3 (July 1996), 5–21.

[15] FLOYD, S., AND KOHLER, E. Internet research needs better models. *SIGCOMM Comput. Commun. Rev. 33*, 1 (Jan. 2003), 29–34.

[16] FLOYD, S., AND PAXSON, V. Difficulties in simulating the internet. *IEEE/ACM Trans. Netw. 9*, 4 (Aug. 2001), 392–403.

[17] GERSHGORN, D. The data that transformed AI research—and possibly the world. Quartz, https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world, July 2017.

[18] 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.

[19] HANDIGOL, N., HELLER, B., JEYAKUMAR, V., LANTZ, B., AND MCKEOWN, N. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2012), CoNEXT '12, ACM, pp. 253–264.

[20] HEMMINGER, S. Network emulation with NetEm. In *Linux conf au* (2005), pp. 18–23.

[21] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[22] JACOBSON, V. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols* (New York, NY, USA, 1988), SIGCOMM '88, ACM, pp. 314–329.

[23] JOHANSSON, I., AND SARKER, Z. Self-clocked rate adaptation for multimedia. Tech. Rep. RFC8298, Internet Engineering Task Force, 2017.

[24] LANGLEY, A., RIDDOCH, A., WILK, A., VICENTE, A., KRASIC, C., ZHANG, D., YANG, F., KOURANOV, F., SWETT, I., IYENGAR, J., ET AL. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), ACM, pp. 183–196.

[25] MOČKUS, J. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference* (1975), Springer, pp. 400–404.

[26] Netflix Open Connect. https://openconnect.netflix.com/.

[27] NETRAVALI, R., SIVARAMAN, A., DAS, S., GOYAL, A., WINSTEIN, K., MICKENS, J., AND BALAKRISHNAN, H. Mahimahi: Accurate record-and-replay for HTTP. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference* (Berkeley, CA, USA, 2015), USENIX ATC '15, USENIX Association, pp. 417–429.

[28] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[29] NTP: The network time protocol. http://www.ntp.org/.

[30] OTT, M., SESKAR, I., SIRACCUSA, R., AND SINGH, M. ORBIT testbed software architecture: Supporting experiments as a service. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities* (Washington, DC, USA, 2005), TRIDENTCOM '05, IEEE Computer Society, pp. 136–145.

[31] PAXSON, V., AND FLOYD, S. Why we don't know how to simulate the Internet. In *Proceedings of the 29th Conference on Winter Simulation* (Washington, DC, USA, 1997), WSC '97, IEEE Computer Society, pp. 1037–1044.

[32] RAMAKRISHNAN, K. K., AND JAIN, R. A binary feedback scheme for congestion avoidance in computer networks. *ACM Trans. on Comp. Sys. 8*, 2 (May 1990), 158–181.

[33] RIZZO, L. Dummynet: A simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev. 27*, 1 (Jan. 1997), 31–41.

[34] ROSS, S., GORDON, G. J., AND BAGNELL, J. A. No-regret reductions for imitation learning and structured prediction. *CoRR abs/1011.0686* (2010).

[35] SCHAPIRA, M., AND WINSTEIN, K. Congestion-control throwdown. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2017), HotNets-XVI, ACM, pp. 122–128.

[36] SHALUNOV, S., HAZEL, G., IYENGAR, J., AND KUEHLEWIND, M. Low extra delay background transport (LEDBAT). Tech. Rep. RFC6817, Internet Engineering Task Force, 2012.

[37] SIVARAMAN, A., WINSTEIN, K., THAKER, P., AND BALAKRISHNAN, H. An experimental study of the learnability of congestion control. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 479–490.

[38] SNOEK, J., LAROCHELLE, H., AND ADAMS, R. P. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (2012), pp. 2951–2959.

[39] Universal TUN/TAP device driver. https://www.kernel.org/doc/Documentation/networking/tuntap.txt.

[40] WHITE, B., LEPREAU, J., AND GURUPRASAD, S. Lowering the barrier to wireless and mobile experimentation. *SIGCOMM Comput. Commun. Rev. 33*, 1 (Jan. 2003), 47–52.

[41] WINSTEIN, K., AND BALAKRISHNAN, H. TCP ex machina: Computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 123–134.

[42] WINSTEIN, K., SIVARAMAN, A., AND BALAKRISHNAN, H. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2013), USENIX, pp. 459–471.

[43] ZAKI, Y., PÖTSCH, T., CHEN, J., SUBRAMANIAN, L., AND GÖRG, C. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, ACM, pp. 509–522.