Configanator: A Data-driven Approach to Improving CDN Performance.

Usama Naseer and Theophilus A. Benson, *Brown University*

https://www.usenix.org/conference/nsdi22/presentation/naseer

This paper is included in the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation.

April 4–6, 2022 • Renton, WA, USA

978-1-939133-27-4

Open access to the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation is sponsored by

Configanator: A Data-driven Approach to Improving CDN Performance.

Usama Naseer
Brown University

Theophilus A. Benson
Brown University

Abstract

The web serving protocol stack is constantly evolving to tackle the technological shifts in networking infrastructure and website complexity. As a result of this evolution, web servers can use a plethora of protocols and configuration parameters to address a variety of realistic network conditions. Yet, today, despite the significant diversity in end-user networks and devices, most content providers have adopted a "one-size-fits-all" approach to configuring the networking stack of their user-facing web servers (or at best employ moderate tuning).

In this paper, we demonstrate that the status quo results in sub-optimal performance and argue for a novel framework that extends existing CDN architectures to provide programmatic control over a web server's configuration parameters. We designed a data-driven framework, Configanator, that leverages data across connections to identify their network and device characteristics, and learn the optimal configuration parameters to improve end-user performance. We evaluate Configanator on five traces, including one from a global content provider, and evaluate the performance improvements for real users through two live deployments. Our results show that Configanator improves tail (p95) web performance by 32-67% across diverse websites and networks.

1 Introduction

Web page performance significantly impacts the revenue of content distribution networks (CDNs) (e.g., Facebook, Akamai, or Google), with studies showing that a 100ms decrease in page load times (PLT) can lead to 8% better conversion rate for retail sites [14, 30]. Yet, uniformly improving web performance is becoming increasingly challenging due to the growing disparity in the network conditions (e.g. bandwidth, RTT) [3, 36, 120, 135] and end-user devices [93, 94, 108, 134]. To address this disparity and improve the quality of experience (QoE), the networking community is constantly developing new protocols and configuration parameters for web servers (AKA, edge servers), e.g., PCC [31], BBR [23], QUIC [51], etc.

The optimal choice of configurations is contingent on the network infrastructure [3, 36, 77, 98, 120, 135, 142], website complexity [20, 21, 95, 134, 136], and end-user devices [1, 94, 108]. Furthermore, innovations along any one of these dimensions will lead to changes to default parameters and new protocols. Although different regions and ISPs have radically different networking infrastructure and mobile devices [1, 93], a majority of CDNs continue to employ a "one-size-fits-all" [49] approach to configuring their edge servers, which results in sub-optimal performance [3, 36, 135] and high tail-latency in certain regions [142].

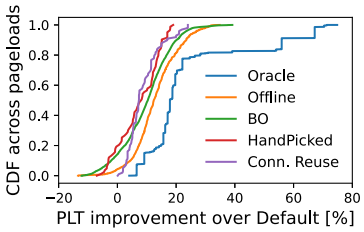


Figure 1: Comparison of various tuning techniques.

1.1 Configuration Tuning Status-Quo

Most attempts to tackle this growing diversity involve manually analyzing the performance of configuration options across different regions [49], devices [1], or websites [110, 135]. While several CDNs expose configuration knobs to their customers [40, 45], it is challenging to take the full advantage of the knobs due to the required manual efforts and the lack of automated learning techniques for effective tuning.

This paper focuses on tuning a broad set of configuration knobs across the transport (e.g., congestion control algorithm) and application layers (e.g., HTTP version) as highlighted in Table 1. Next, we illustrate the challenges and benefits of dynamically tuning network configurations.

Challenges in tuning stack: In Figure 1, we illustrate the difficulty of tuning configurations by comparing page load time (PLT) of popular websites, when configured using popular tuning techniques (setup explained in § 2.2). Specifically, *Bayesian Optimization* [104] (e.g., CherryPick [4]) a statistical

technique used for tuning systems configurations [4, 32, 75, 131], operator *hand-tuned* configurations (discussed in § 2), *TCP connection reuse* a traditional optimization (discussed in § 6.5), and a closed-loop *offline-learning* technique. We compare their performance against two baselines: *optimal* configuration discovered through an exhaustive brute-force search, and *default* configurations for Linux and Apache (Table 1).

Hand-tuned configurations are manually selected and are thus, coarse-grained. While they out-perform the *default* at median, they fail to provide optimal performance across varying network conditions and may even lead to performance degradation for some networks. TCP connection reuse only optimizes a subset of knobs (e.g., initial congestion window) and is unable to take full advantage of the diverse network stack knobs. Bayesian optimization aims to quickly discover "good" configuration. While fine-grained, this approach is relatively static and does not re-evaluate old choices, and is thus unable to adapt to network dynamics [78]. We observe the effects of this rigid behavior with wildly varying tail performance. Lastly, we explore an offline model which learns on traces from prior days and applies the learned model on connections for the next day. Offline modeling is fine-grained but with limited dynamicity: the trained model is unable to react to real-time issues. Unfortunately, due to the high dimensionality of the Internet's dynamics, these real-time issues are the norm, not the exception [67, 69, 78]. We observe in Figure 1 that offline performs closest to the optimal but still falls short because of its inability to react in real-time.

Our brief analysis of tuning approaches highlights the need for a dynamic, fine-grained approach to tuning configurations.

1.2 Configanator

In this paper, we eschew the notion of a homogeneous approach to tuning web server configurations and instead argue for a "curated" approach for configuring on a per-connection basis. In particular, we argue that edge servers should be configured to serve each of the incoming connections with the optimal protocols and configuration parameters, e.g., a web server may employ Cubic in favor of BBR when serving a low bottleneck buffer connection [111, 114]. To this end, we argue for a simple but robust server architecture that introduces flexibility into the network stack, enables reconfiguration, and systematically controls configuration heterogeneity. We also introduce a contextual multi-armed bandit based learning algorithm, an embodiment of domain-specific insights, which tunes configuration in a principled manner to find optimal configurations in minimal time. Taken together the design and the learning algorithm, our system, *Configanator*, enables a CDN to systematically explore heterogeneity in a dynamic and fine-grained manner while improving end-user performance. The design of Configanator faces several practical challenges:

- Network dynamics: network may change every few minutes [67, 83, 146] and thus requires continuous learning.

Layer	Protocol Options	Default	Example parameter
Transport	congestion_control (CC)	Cubic	BBR, Cubic, Reno
	initial congestion window	10 MSS	Integer (1, 4, 30)
	slow_start_after_idle	1	boolean {true, false}
	low_latency	0	boolean {true, false}
	autocorking	1	boolean {true, false}
	initRTO	1s	decimal (0.3,1)s [59]
	pacing (fair-queue)	0	boolean {true, false}
	timestamps	1	boolean {true, false}
	wmem	{4096}B	{163840}B
Web App	HTTP Protocol	1.1	1.1, 2
	H2 push	On	On, Off
	H2 max header list size	16384B	Integer values
	H2 header table size	4096B	Integer values
	H2 max concurrent streams	100	Integer values
	H2 initial window size	65535B	Integer values $< 2^{31}$
	H2 max frame size	16384B	Integer values $< 2^{24}$

Table 1: Web stack configuration parameters.

- Non-Gaussian noise: CDNs focus on improving tail latency [27, 53, 145] which is often caused by non-Gaussian processes (e.g., last-mile contention [125], mobile device limitations) and are difficult to model.
- High-Dimensionality: Content personalization, diverse devices [94, 134], and last-mile connections [125] introduce high dimensionality that limits the efficacy of offline closed-loop approaches [67, 68].
- High data cost: Generating data for learning requires testing configurations and may disrupt user's performance. Hence, the negative impact on users must be minimized.
- Limited flexibility: Linux kernel and modern web servers lack the flexibility to tune configurations on a per-connection basis, thus requires enhancing the traditional networking stack.

The key insight of Configanator is to simultaneously operate in two modes depending on the "quality" of the performance model. Essentially, Configanator intelligently selects samples that speed up model convergence, then at steady-state it transitions into a greedy-mode that stochastically samples points to iteratively improve performance. Configanator further clusters similar connections together and samples across clusters to amortize the cost of exploration.

Configanator uses a contextual multi-armed bandit [133] designed explicitly to continuously converge to an optimal (or near-optimal) configuration within a minimal number of exploration steps. Our ensemble fuses the stateful exploration of Gaussian-bandit with the non-determinism of Epsilon-bandit, enabling informed exploration of the configuration space while randomly re-sampling old configurations. The re-evaluation of data samples enables Configanator to directly tackle non-Gaussian noise within the domain. The data collected by the ensemble is encoded in a decision tree – which enables quick and easy classification but is also amenable to automatic generation of rules for a CDN's web server.

To demonstrate the benefits, we conducted large-scale simulations and live deployments. We used datasets from a *GlobalCDN* and public datasets from CAIDA [22], MAWI [8], Pantheon [142] and FCC [41]. Our simulation results show that Configanator provides 32-67% (up to 1500ms) improvement in the PLT at tail (p95) across the different traces. Given the

Layer	Option	Top configs. in N.A. (cross-CDN)	% of CDNs configuring differently across regions	Example of observed cross-regional difference
Web App	HTTP version	H1.1(44.3%), H2(55.7%)	4.7%	N.A. H2 -> Asia H1.1
	Max header list size	16384 (100%)	0%	None
	Header table size	4096 (100%)	0%	None
	Max concurrent streams	100 (44%), 128 (56%)	1%	N.A. 100 -> EU 128
	Initial window size	65536 (71%), 65535 (15%), >1M (14%)	1.9%	N.A. 1048576B -> Asia 65535B
	Max frame size	16,777,215 (81%), 16384 (19%)	0%	None
Transport	ICW	{10 (62%), 4(20.5%), 24(5.3%)} MSS	6.9%	N.A. 24 MSS -> Asia 10 MSS
	initRTO	{0.3(9.2%), 1(82.6%), 3(8.2%)} sec	2.3%	N.A. 3s -> EU 1s
	RWIN	{29200(57.4%), 14600(8.2%), 42780(6.8%)} bytes	3.6%	N.A. 29200B -> Asia 12960B

Table 2: Heterogeneity in configs. across 5 regions

recent arms race by CDNs to improve web performance, we believe that Configanator's modest improvements will result in significant revenue savings [14, 19, 30, 103]. Please refer to the project website [1] for the related resources.

2 Empirical Study

Next, we analyze CDNs to determine the current extent of configuration tuning (§ 2.1) and quantify its implications (§ 2.2).

2.1 Fingerprinting web configurations

We aim to understand if modern CDNs employ homogeneous configurations, as suggested by anecdotal evidence, or heterogeneous configurations to tackle diversity in the Internet's ecosystem. To this end, we developed a tool to infer and fingerprint a web server's [49, 92] application/L7 and transport/L4 layers configuration parameters by actively probing the servers and inspecting the packet headers and their reaction to emulated network events (e.g., packet loss). Please refer to Appendix A for more details about the tool. Using the tool, we fingerprinted the configurations for the Alexa top 1k websites from five different regions (North America (N.A), South America, Asia, Europe, and Australia), and present the results in Table 2. We use N.A configurations as the reference point and compare the observed configurations along two axes:

Observation 1: Heterogeneity across CDNs: In Column 3 (cross-CDN), we observe that different CDNs use different configurations in N.A. While some of the heterogeneity can be attributed to differences in the default values for different OSes, we observe that CDNs do use non-default values, e.g., amazon.com uses an ICW of 24 MSS in N.A.

Observation 2: Homogeneity within a CDN: In Column 4 (cross-region), we observe that only a small number of CDNs tune their network stack to account for regional differences, i.e., use different configurations in N.A. than the other regions. The highest amount of tuning occurs at L4, with 6.9% of the CDNs tuning the ICW differently in N.A. than in other regions, e.g., 24 MSS in N.A. but 10 MSS in Asia for amazon.de.

Takeaway: Taken together, these observations indicate that while individual CDNs perform modest tuning, most do not tune finely enough to account for regional diversity. In fact, only a small set of CDNs configure differently across regions.

[1] https://systems.cs.brown.edu/projects/configtron/

2.2 Implications of Configuration Tuning

Next, we quantify the benefits of dynamically tuning a web server's networking stack by conducting a large scale study in our local testbed. We emulate a wide range of representative networks (extracted from real-world traces [8, 22, 41, 142]) and perform an exhaustive, brute-force search of configuration space (detailed description of the traces is provided in § 6.1). Table 1 lists the set of configurations, with default settings for TCP and HTTP taken from the Linux transport stack (kernel 4.20) and Apache (v2.4.18), respectively. In each trial, the server iteratively selects a configuration from the possible configuration space, a representative network is emulated using NetEM [54], and the PLT of a randomly selected website from Alexa Top-100 (locally cloned on the server) is measured five times. The *optimal* configuration is defined as the one that results in the lowest PLT for a specific network and website.

Figure 1 explores the implications of using sub-optimal configurations, by comparing optimal and default configurations for pageloads across diverse networks and websites. We observe that there is ∼18% PLT improvement at the median (over 70% at tail) when optimal configurations are used over the default. While the number may appear small, they can result in tremendous revenue improvements [14, 30], and more in the developing regions where CSPs are investing heavily to improve network [37, 80]. We observe the highest reconfiguration benefits for low bandwidth, high RTT/loss regions, representative of developing region networks.

Next, we analyze congestion control measurements across different regions from Pantheon [142]. We observe that emerging protocols, e.g., BBR, PCC, or Remy, which use probing or ML to improve performance, do not provide uniformly superior performance. In particular, we observed that in many situations BBR is suboptimal, performing 3X to 10X worse than the optimal congestion control. Moreover, no congestion control is optimal for more than 25% of the networks tested, and the median congestion control is optimal for only 6% of the networks.

3 Configanator's Algorithm

Tuning network configurations to maximize the web performance for diverse networks and end-users presents a complex learning problem. Next, we formulate the problem and present a domain-specific ensemble to address the challenges.

Problem Formulation: Given a set of networking configurations ($C=\{c_1, c_2 ... c_n\}$), network conditions (N

$= \{n_1, n_2 ... n_n\}$), devices ($D = \{d_1, d_2 ... d_n\}$), websites ($W = \{w_1, w_2 ... w_n\}$) and a function, *f()*, that maps a website, network condition, device, and configuration to a metric of web page performance (e.g., PLT or SpeedIndex). Note that, $f(c_i,n_i,d_i,w_i)$ returns the web page performance metric value for applying configuration c_i to a user device d_i loading website w_i in network n_i. In this paper, we use PLT as the metric for web page performance and can be easily replaced with other metrics. Our goal is to solve Eq. 1 and find a configuration ($c*$) that minimizes *f()* for a given combination of n_i, d_i and w_i.

$$\underset{c*}{\operatorname{argmin}} f(c*,n_i,d_i,w_i) = \{f(c_i,n_i,d_i,w_i) | \forall c_i \in C\} \quad (1)$$

Solving the black-box function f() requires exploring sample space. Two possible exploration algorithms are:

- *Brute force* [2] which tests each possible configuration one by one until the entire space is explored.
- *Bayesian optimization* (BO) [16, 104] is a principled global optimization strategy that uses a prior probability function to capture the relationship between the objective function (Eq 1) and the observed data samples. BO models f(c,n,d,w) as a Gaussian process (GP) [16]. GP is a distribution of candidate objective functions and is used to select the next promising point (c*) which is then evaluated on a connection. GP then updates its posterior belief by adding the new observation f(c*,n,d,w) to the set of seen observations. With every new observation, the space of possible candidate functions gets smaller and the prior gets consolidated with the new evidence.

Challenges: Both approaches are sub-optimal for our use-case due to several reasons: (1) non-stationary network conditions [10, 67, 69, 146] (network conditions change every few minutes), (2) BO assumes that data is noise-free or only has Gaussian noise [118], and non-Gaussian noise (tail latency can not be modeled by a Gaussian process [78]) disrupts the estimation of next candidate sample and is observed to impact BO's hyper-parameters (e.g., threshold on expected improvement for next sample to stop the exploration), (3) costly data collection (collecting data requires testing on end-users which can impacts PLT and revenue), (4) data scarcity (testing on individual users requires each user to generate a tremendous number of connections but a user may only visit the site a few times).

Intuition: The intuition behind Configanator's algorithm is to decompose the model building into two phases: (i) an initial phase during which the search should be directed to speed up the process and build a good (not perfect) model, and (ii) a steady-state during which the search should be more stochastic to iteratively improve the model and tackle non-Gaussian noise. Building on these insights, Configanator leverages a combination of clustering, an ensemble of bandit-techniques, and ML to address the aforementioned challenges. Specifically, clustering is used to group connections based on their network and device similarity (called *Network Class*) and aggregate observations across similar connections to address data scarcity. The use of a contextual multi-armed bandit [133] enables Configanator to explore configurations and continuously collect data samples to learn and tackle dynamic client-side conditions in a balanced and online manner. To generalize observations across the connections, a Decision Tree is trained for efficient inference.

3.1 Domain-Specific Multi-Armed Bandit

Configanator's learning algorithm consists of a contextual multi-armed bandit [76, 84, 133] with three arms:

- **Exploration Arm-1 (Gaussian process [104, 112]):** The Gaussian process (GP) bandit [4, 73] uses an acquisition function to perform a directed search to quickly discover a "good" (might not be optimal) solution when no information exists for a Network Class (NC). There are multiple acquisition functions available [16] and we use *Expected Improvement (EI)* [112] because of its well-documented success [4, 32, 47]. This search process includes two terminating conditions: a threshold on EI and minimum of number of data points to explore. For non-continuous configurations (e.g., HTTP version), we encode them into a number to discretize the space[2]. To account for performance differences between websites and NCs, the GP-arm is composed of a collection of GP models, one for each unique website and NC combination (Appendix D).
- **Exploration Arm-2 (Epsilon-bandit [128]):** The Epsilon-bandit randomly re-samples the data points to overcome issues endemic with the Gaussian process (and Bayesian Optimization in general), e.g., non-stationarity of mean performance. The network operator bounds the random exploration by defining a parameter, ε, that controls the trade-off between speed of exploration and the impact on end-user QoE. A high ε improves exploration but results in a negative impact on clients' QoE due to constantly changing configurations.
- **Exploitation Arm (Decision Trees [107]):** The exploitation arm uses ML-powered prediction to model the data collected through the exploration arms. We evaluated several techniques including Support Vector Machines, Decision Trees (D-Trees), and Random Forests. We found that the D-Tree hits the sweet spot, providing comparable accuracy to the other models while being efficient enough to build and update at scale. The D-Tree encompasses all websites and NCs to learn across websites and networks. Leveraging the config-performance curves collected by underlying exploitation arms, a single D-Tree model is trained for the "good" configuration found so far for each website/NC pair, and the D-Tree maps {website, device, network/AS characteristics} to their optimal configuration.

Context-based arm switching: Configanator constantly switches between the arms based on the NC's "context" which is defined as the quality of the GP-model for the website/NC. It operates in two modes: (i) *Bootstrap*, when no information exists for a website or NC, the context is empty and the GP-arm is used to explore the configuration space in a principled manner until the acquisition function (EI) indicates

[2] GPyOpt [101] supports mixed (continuous/discrete) domain space [102].

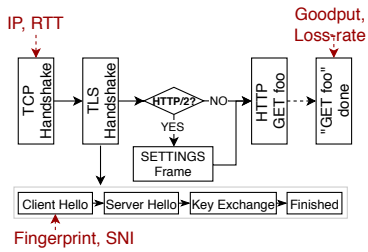


Figure 2: Connection features.

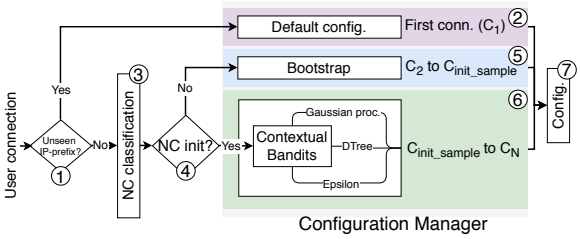


Figure 3: Learning framework workflow.

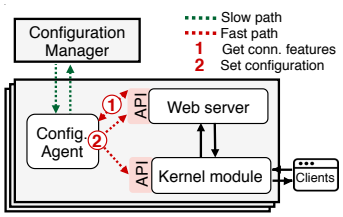


Figure 4: System architecture.

that a good configuration is found, (ii) *steady-state*, when information from the GP-arm indicates "good" configurations, Configanator uses either the epsilon-bandit to further explore the configuration space, or the exploitation arm (i.e., D-Tree) to leverage best configurations. Note that, random exploration through epsilon-bandit continues after EI threshold is met.

3.2 Discovering Network Classes

Configanator extends on observations from prior studies [67, 89] and classifies homogeneous connections into *Network Classes (NC)* with the intuition that similar connection characteristics lead to identical optimal configurations.

Design Goals and NC Features: The ideal NC-clustering should (i) create a small number of clusters, each with a large number of connections to amortize the cost of explorations, and (ii) all members of a cluster should have near-identical profiles. The two goals inherently contradict: the greater the number of entities in an NCs, the higher the probability that the NC contains entities with diverging performance. The second goal is further complicated by the sensitivity of a configuration's performance (e.g., PLT) to a myriad of factors in the end-to-end connection. To this end, we use network characteristics (bandwidth, latency, loss rate), AS information (ASN, geo-location), and device type as the basis for measuring similarity.

Capturing NC Features: To enable Configanator to effectively tune both the transport and HTTP layers, we must identify all features during the TCP handshake before the HTTP version is negotiated through ALPN [60]. If we identify features after HTTP negotiations, then tuning the HTTP layer would require renegotiation and hence incurs latency penalty. In Figure 2, we highlight the features collected during specific phases of the connection: (1) During the TCP handshake, we capture RTT, IP-prefix, and ASN/geo-location[3]. (2) During the TLS handshake, we apply TLS fingerprinting techniques [5, 18, 70, 127] on the TLS *Client Hello* to perform device identification and capture device features (accuracy evaluated in Appendix B). Note that, most operators already employ TLS fingerprinting for security purposes [6, 61, 126] and is also supported by major web servers [29]. We use the *Server Name Indication (SNI)* in the *Client Hello* to determine the website hostname which is one of the input features for the learning framework. (3) For goodput and loss rates, features that cannot be captured during handshake, we build and use a historical archive of these network characteristics.

Network Classification: Clustering can be done using conventional techniques, e.g., K-means, hierarchical, or domain-specific techniques [17, 38, 105], e.g., Hobbit [74] or, CFA [67], or using CDN state of the art [26, 87, 119, 139, 140], e.g., latency-based groups [26, 119, 139]. Although Configanator can incorporate any of the aforementioned techniques, our prototype uses "K-means" clustering because of its simplicity. Configanator empirically selects the smallest K (i.e., the number of classes) that bounds the spread of performance within each NC by a predefined limit[4](evaluated in § 6.2 and Appendix D).

3.3 Configanator Workflow

Figure 3 presents the end-to-end workflow. Default configuration is initially used for a newly-seen IP-prefix (①, ②) due to the lack of information about its goodput and loss-rates. For any subsequent connection from the IP-prefix, the recorded, as well as the actively collected features, are used for NC classification (③). If the network, AS and device characteristics do not fit into an existing NC, a new NC is created (④) and the next *init_samples* connections for the respective NC are used for bootstrapping (⑤) its empty context. When the respective NC is bootstrapped, the multi-armed bandit uses the actively and passively collected features, as well as the requested website, for determining the context and alternates between the arms (⑥). The connections are correspondingly tuned (⑦) and the resulting performance metrics are fed back into the models to help refine their classifications and improve accuracy. Due to the computationally intensive nature, Configanator builds/updates NC clusters in the background and uses the already-built clusters for real-time classification.

4 Architecture

Our re-architected web server consists of four components (Figure 4): The *HTTP server application* [39, 97, 121, 132] operates as it does today: serves content and collects performance metrics for each connection. The *Configuration Manager* runs the learning algorithm on the telemetry collected from the web servers. The *Configanator-API* abstracts vendor-specific

[3]Captured using end-user's IP and publicly available data (RouteViews for AS [17], MaxMind for geo-location [62])

[4]Controlled by *NCSpread* knob in simulator (Table 4 in Appendix).

configuration details and provides a uniform interface for configuring web server's network stack parameters. A *Configuration Agent* runs on each web server and uses the information received from the Configuration Manager to configure the connections through the Configanator-API.

Adopting this architecture in an incrementally deployable manner is practically challenging. The configuration parameters are exposed in an ad-hoc manner, e.g., tuning transport configuration requires *IOCTL* and *setsockopt*, while tuning HTTP requires changes to application code and enhancements to the ALPN protocol. Additionally, most CDNs use well-established code bases and exposing the configuration interfaces required by Configanator should incrementally build on the existing code.

4.1 Configanator-API

The *Configanator-API* presents a uniform interface over the web server's serving stack thus abstracting away OS and web server specific details. This simplified interface enables the *Configuration Agent* to easily tune the network stack, without having to understand vendor-specific details or implications.

Transport tuning: Unfortunately, the traditional kernels only expose and provide flexible reconfiguration for a subset of TCP's parameters. In particular, some parameters (e.g., ICW) can be configured on the connection level, while others can only be configured on a global scale (e.g., tcp_low_latency). Using Configanator at a coarser granularity, either limits the type of supported connections on a machine or limits the configuration space. There are several options to address this issue ranging from user-space TCP/IP stacks [35, 65, 106], kernel modules, eBPF programs, to leveraging virtualization. We opt for a kernel module-based design over virtualization approaches because hosting a single configuration per VM introduces significant overheads.

HTTP tuning: HTTP version and H2 settings are determined through Application Layer Protocol Negotiation (ALPN) [60] in TLS handshake and H2 SETTINGS [12] frame, respectively. Given the requirement for per-connection tuning, we augment the ALPN and the H2 SETTINGS frame code to enable fine-grained control over these configurations. In particular, Configanator configures these settings by restricting the options presented in the server advertisement to the configuration setting being tuned, e.g., to set the HTTP protocol to H2, we limit the "ALPN next protocol" field in TLS *Server_Hello* to just H2. Similarly, we restrict the options in the SETTINGS frame to configure HTTP/2 settings.

Tuning Workflow: Configanator-API tunes both the TCP and HTTP version during the TLS handshake: after receiving the *Client Hello* from the end-user and prior to sending the *Server Hello*. This is the perfect location to tune because (1) the complete feature set required to determine a connection's NC and configuration can be captured at this point, and (2) the server is yet to finalize the HTTP protocol, which the ALPN selects in *Server Hello*, thus enabling us to configure the HTTP version. We note that at this phase of the connection, the TCP state machine is in its infancy because the sender has not sent any data, and thus virtually no significant state is lost when we change the congestion control algorithm or settings.

4.2 Configuration Agent

The *Configuration Agent* is the glue logic between the *Configuration Manager* and *Configanator-API* — it collects the connection features, uses rules provided by the Configuration Manager to make configuration decisions, and configures them using the *Configanator-API*. We select a proactive approach, where the *Configuration Manager* constantly pushes NC and configuration mappings to the *Configuration Agent* which caches them locally. Further for an unseen IP-prefix, *Configuration Agent* uses the default configuration, until the *Configuration Manager* finds a better mapping.

4.3 Configuration Manager

The manager runs in a centralized location, e.g., a data center or locally in a Point of Presence (PoP), with the implications later explored in Appendix F.9. It is charged with running the learning algorithms (§ 3), network classification models (§ 3.2), and disseminating the configuration maps to the *Configuration Agents*' cache. The *Configuration Manager* disseminates and collects data from the *Configuration Agents* using distributed asynchronous communication. For the NC and configuration maps, *Configuration Manager* broadcasts to all *Configuration Agents*, whereas for reporting performance data and for making one-off-request for configuration maps, the *Configuration Agents* use unicast.

5 Prototype

The implementation highlights of the prototype are as follows:
Configanator-API: partly resides within the kernel (as a module) and partially resides in user-space in the form of additions to the web server code (in our case Apache). The components within the kernel allow us to tune the transport, while the user-space allows us to tune the HTTP layer.

The kernel module reuses functions provided by kernel's congestion controls through the *tcp_congestion_ops* interface and tunes fields in appropriate structs (e.g., *inet_connection_sock*). For tuning globally-defined knobs at a per-connection level (e.g., *tcp_low_latency*), we leveraged kernel patches [34,55] to define and reference them from *tcp_sock* struct. The user-space component within Apache code tunes HTTP version in Apache and its design is generalizable to other servers that use OpenSSL. OpenSSL library is used by most web server implementations and allows web servers to register a *SSL_CTX_set_alpn_select_cb* [44] callback to modify ALPN decisions. To tune HTTP version, we register a call back which looks up the HTTP version to use for a connection and restricts the ALPN options advertised to the one specified by the Configuration Agent. For H2 settings, we modify the Apache H2

module to dynamically select the configurations while sending the SETTINGS frame. The user-space agent also generates the TLS fingerprint for device identification. We use JA3 [70] for TLS fingerprinting. In both our testbed experiments and in the live deployments, we use the ALPN-centric approach which modifies protocol options presented in the advertisements.

Configuration Agent: is user-space code and is implemented in 492 LoC of C++ code. The agent updates TCP and HTTP settings via the Configanator-API. This component also parses Apache's logs for measuring network characteristics. For measuring the PLT, the web server injects a simple JavaScript into the webpage to measure the navigation timings.

Configuration Manager: is developed in 1435 LoC (Python). It uses SciLearn [116] for D-Tree and GPyOpt [101] for the Gaussian Process. For communication with the Configuration Agents, we use ZeroMQ [144]. For D-Tree, we use SciLearn's CART algorithm with the following configuration: (i) entropy for the information gain, (ii) set the minimum number of leaf nodes to 80, (iii) set the minimum number of samples needed for the split to 2, and (iv) do not limit the depth of tree. For Gaussian process, we use init_sample=4, min_sample_tested=7 and EI=8% thresholds. We tested a range of these hyper-parameters settings [5] and selected the ones resulting in the highest accuracy. Following [4], we tested EI threshold in 3-15% range and selected 8% for its best trade-off between accuracy and search cost. For controlling the "K' for NCs, we use a *NCSpread* threshold of 5% (Appendix D).

6 Evaluation

We evaluated Configanator through a large-scale, trace-driven simulator using real-world traces, and live-deployments (§ 7). The simulation enables us to understand the system behavior under dynamic conditions, as well as analyze the implications of individual design choices.

6.1 Large Scale Trace Driven Simulations

Datasets: To simulate client activity, we use data from five sources: (i) *GlobalCDN* comprises 8.2M requests sampled from web and video services from 3 GlobalCDN PoPs (two in N. America, one in Europe) for a duration of 6 hours. Each request is a client fetching an object (e.g., web object, video chunk, etc.) and contains user information (IP prefix, ASN, etc.), observed server metrics (goodput, RTTs, loss rates etc.), CDN logs (e.g., user to edge PoP mapping [26, 119]) and performance metrics (time-to-last-byte). (ii) *CAIDA* [22], packet traces from the Equinix data-center in Chicago (in 2016). (iii) *MAWI* [8], packet traces from the WIDE backbone in Japan (in 2017). (iv) *FCC* [41], a U.S. nation-wide home broadband dataset. (v) *Pantheon* [141, 142], a data set of client sessions across different regions.

[5] (e.g., entropy vs Gini impurity for information gain, number of leaf nodes ranging from 50 to 500, ID3, C4.5, and CART for D-Tree)

Generating client sessions: We use our traces to characterize the network conditions of real-world users. CAIDA and MAWI traces are captured at a vantage point between the client and server and we measure the goodput, RTT and loss rate by sequence-matching the data packets with their ACKs [6]. GlobalCDN[7], FCC and Pantheon datasets include the end-to-end network characteristics between a client and server. We model a client session as a time series of bandwidth (goodput), latency and loss rate measured between a pair of end-points.

Configuration Rewards: To avoid the pitfalls of trace-driven simulations [11], we decouple the modeling of configuration rewards (i.e., PLT calculation) from the process of generating client sessions. Our testbed comprises a cluster of 16 Linux servers (kernel 4.20), divided evenly to act as server (Apache) and clients (Chromium [50]). Our control over the machines and network enable us to set arbitrary server-side configurations (from Table 1) and emulate the bottleneck link to match the measured goodput, latency and loss rates from the datasets (using NetEm, TC [54]), with buffer-size set to Bandwidth Delay Product (BDP). To isolate the impact of network and configuration on PLT, caching (server or browser) is disabled and each server serves a single client (no resource contention). Using this testbed, we exhaustively measure the PLT for all combinations of configurations (Table 1). For each {network condition, configuration} pair, each website is loaded multiple times with the browser[8]. The final results are stored in a large tensor that maps {network condition (goodput, RTT, loss-rate), configuration, website} to PLT – called the *PLT-Tensor* comprising data from the pageloads in the testbed.

Simulator (Virtual Browser): Leveraging the client sessions and the PLT-Tensor, the simulator simulates the client's browsing behavior and interaction with Configanator as follows (visualized in Appendix F.1): (i) website[9], user information (e.g., IP) and session characteristics are taken as inputs, (ii) the learning framework determines the appropriate configuration for a connection, and (iii) pageload is simulated by using the PLT-Tensor to determine PLT for the client given the selected configuration. The simulator feeds the PLT back to the learning framework to complete the feedback loop.

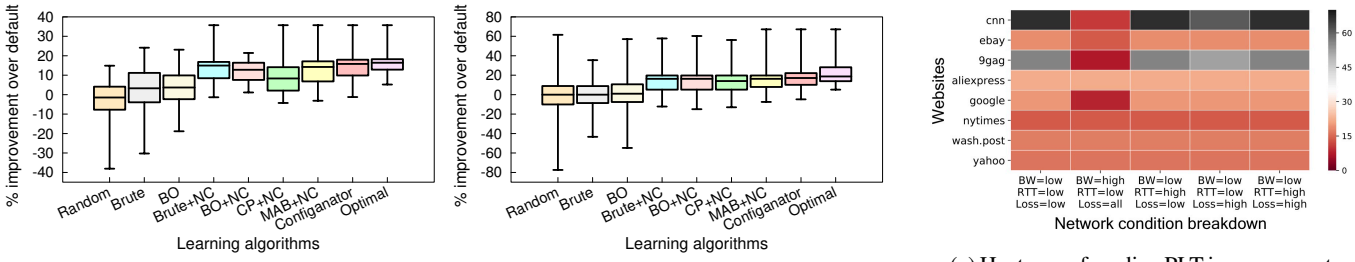
PLT is sensitive to a myriad of features, ranging from dynamic network conditions at different time-of-the-day [67], user devices, to inherent variability. The session time-series captures the network dynamics and the testbed isolates the impact of network conditions on configurations. Table 4 lists the set of knobs we leveraged to test various realistic design choices, e.g., *NCSpread* to test various "K" sizes, *PerfMemory* to test the impact of PLT variability, etc. To account for the other factors like user devices, we conducted a scaled-down

[6] Over a 5-second window (tunable through *ChunkSize* parameter in the simulator), e.g., data ACKed in 5s is used to measure goodput between vantage point/user. Further, we ignore duplicate ACKs while measuring RTTs.

[7] Measurements are on a per-request granularity. For multiple readings within the 5s window, we aggregate and use the median value.

[8] We repeated each measurement 5 times, similar to [135].

[9] We iteratively load every website from our corpus for a given session.



(a) MAWI traces (b) GlobalCDN traces (c) Heatmap of median PLT improvement in different networks (GlobalCDN)

Figure 5: Benefits of Configanator (box whiskers show 5th and 95th percentiles).

experiment on a CDN and tested different configurations for the real-world, diverse user devices (results in § 7.1).

Alternate algorithms: We evaluate against 8 algorithms:
(i, ii) Brute-force (Brute, Brute+NC): explores individual configurations, in an online manner, until all are explored and uses the best one (i.e., lowest PLT) for subsequent connections. *Brute* learns at the granularity of individual clients (i.e., unique IP) while *Brute+NC* clusters clients into Network Class (NC), and thus learning is spread across each NC.
(iii, iv, v) Bayesian Optimization (BO, BO+NC, CherryPick+NC): Bayesian Optimization is used to explore the configuration space and the best-explored option is exploited once BO-specific thresholds are met (§ 5). *BO* learns per client and *BO+NC* learns on a user group (NC) granularity. *CherryPick+NC* is similar to *BO+NC* but with hyper-parameters specified in [4].
(vi) Multi-armed Bandit (MAB+NC): uses traditional MAB with a weighted epsilon-greedy agent [133]. Each arm of the bandit is a different configuration, tested on NC granularity.
(vii) Random: Randomly selects a configuration in each trial.
(viii) Optimal: An oracle suggests the optimal parameters for a session by offline brute-force , i.e., PLT is calculated for the entire configuration space for each session offline and the configuration with the lowest PLT is used. This process is repeated for every session and puts an upper bound on improvement.

6.2 Effectiveness of Configanator

Figures 5a, 5b present the improvement in PLTs over default configurations for the different algorithms. The box plots compile data across the website pageloads for the client sessions in the respective trace. Configanator outperforms all alternatives at median and tail, improving p95 PLTs by 67% for GlobalCDN (1500ms), 36% (1100ms) for MAWI, 32% (610ms) for FCC, 48% (640ms) for CAIDA and 57% for Pantheon (850ms). Unlike Default, while Brute and BO apply different configurations to users, they assume that the network remains static and are unable to adapt to fluctuations. Moreover, due to its inability to adjust to fluctuations, BO often explores over 90% of the space without achieving the target EI, behaving similarly to *Brute*. Brute+NC, BO+NC and CherryPick+NC improve over the prior by amortizing the costs of learning but fail to adjust to non-Gaussian variations. Although *MAB+NC* is able to handle non-Gaussian noise, it explores/exploits on a per-NC basis and, due to the lack of a cross-NC exploitation arm (Configanator's DT), *MAB+NC* falls short in its ability to apply patterns learnt across NCs.

As Configanator continuously learns and tests new configurations in an online fashion, a 'bad' configuration may be tested during the exploration phase and may lead to performance degradation. This behavior contributes to the worse PLT than Default for the p5 pageload in Figures 5a, 5b. A breakdown of the performance degradation and its causes are presented in Appendix F.8.

Dissecting Performance Improvements: Next, in Figure 5c, we analyze performance breakdown for a subset of websites according to the networking conditions used in prior work [135]). We make two observations: (i) Improvements tend to be higher in low bandwidth, low to high RTT/loss networks (typical for developing regions) with a median value of 14-67% compared to 10-25% for high bandwidth. We postulate that this trend is an outcome of the higher focus on developed region networks (typically high bandwidth, low RTT/loss) for the default configuration selection [33]. We observe a similar trend across our traces: GlobalCDN, MAWI and Pantheon traces (p95 RTT in 100-180ms) tend to show higher improvements than FCC and CAIDA (US-based, ∼60ms p95 RTT). (ii) the websites with highest benefits tend to be content-rich, e.g., 9gag.com and cnn.com observe >45% and >60% improvement, respectively, for all low bandwidth networks.

6.3 Benefits of Learning Ensemble

Next, we analyze the convergence for the top-3 algorithms from § 6.2 to focus on the aspects of Configanator that lead to better performance. We further split Configanator into two versions to analyze the benefits of its bandits: "NoGP" lacks GP and guided exploration, while "NoDT" lacks the decision tree.

Figures 6 plots the median distance from optimal across all NC and websites, for the first 500 update iterations. The observations are: (i) As data is gathered, Configanator performs better than others because of its ability to blend the benefits of both GP and DT – essentially efficient exploration and effective exploitation (iterations 3-10). Brute+NC exhaustively explores the complete space before converging to a choice, while MAB+NC exploration lacks the guided nature of acquisition

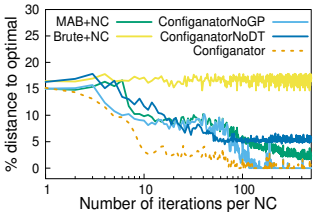


Figure 6: Cold-start convergence to optimal across NCs.

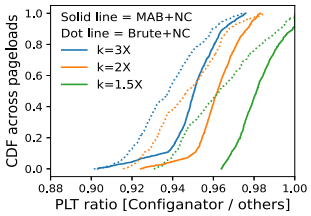


Figure 7: Impact of number of NCs (K) on performance.

function. (ii) Eventually, with sufficient data Configanator-NoGP is able to use the decision tree's predictive power to achieve near ideal performance (iterations 100+). Although MAB+NC gets within 2-3% of optimal for these iterations, it still needs more iterations to reach the optimal. (iii) While NoGP perform comparably for median at 200+ iterations, performance at the tail is **still** different (Appendix F.4).

6.4 Impact of Network Classes

Impact of Number of NCs: Next, we evaluate the impact of our clustering configuration (i.e., *NCSpread*) and analyze how the cluster size impacts performance. Intuitively, *NCSpread* bounds the performance variance within a cluster and has a direct impact on the number of clusters, or 'K'. Given a *NCSpread* value, the simulator performs a brute-force search to determine the smallest K that yields the threshold. We tested three scenarios with K inflated to $\{1.5, 2, 3\}$ times the baseline value (Figure 5 experiments). The inversion from *NCSpread* to K and its implications on modeling accuracy are further discussed in Appendix D.

Figure 7 plots the ratio of Configanator and {MAB+NC, Brute+NC} PLT across the pageloads in GlobalCDN trace (<1 when Configanator outperforms). We observe the performance gap between Configanator and others increases with the K size. Although the large K results in a higher number of tighter NCs with lower performance spread within their constituents, it leads to an overall increase in exploration steps for MAB+NC and Brute+NC, as these algorithms explore the individual NCs independently. Further, the individual NC's best-found configuration is exploited for a narrower set of connections due to a lower number of connections in each cluster as compared to the case when K is small. On the other hand, the DT-arm in Configanator builds on the data collected for *all* NCs (§ 3.1). As soon as Configanator switches to DT-arm fairly early (Appendix F.3), it is able to exploit the best-found configuration for a wider audience, irrespective of the NC boundaries. The higher degree of exploration required by MAB+NC and Brute+NC makes their performance sub-optimal for the NCs with a smaller number of connections. Moreover, this can also lead to performance problems for tail connections, who are often in smaller NC due to their divergent network and device characteristics.

Impact of Size of NC (# of connections): Configanator aggregates network measurement across similar connections and assumes homogeneity within an NC. Though an NC with a small number of users may lead to a smaller number of connections to learn from, it also favors the system as connections in the respective NC are strictly homogeneous. Next, we explore the impact of this bias on our results. We divide the NCs based on their unique number of IP prefixes and compare the PLTs observed for the individual prefixes with the NC's global PLT, i.e, median across all the prefixes in the NC. For two of such divisions, Table 3 presents the PLT comparison across the prefixes in NC groups. Compared to the $<=5$ group, i.e., NCs with a small number of distinct prefixes, where performance for most prefixes matches the global one; $>=30$ group shows more varying performance (e.g., lower than global PLT for the p25 prefix). However, we observe that the presence of larger NCs does not drastically impact Configanator as performance for most of the prefixes is still on par with the global one. For the tail prefixes that performs poorly as compared to the global PLT for the $>=30$ group, Configanator overfits the best-found configuration for the NC majority to the tail prefixes, and is observed to still outperform the *Default* (row 4 and 6 in Table 3).

6.5 TCP Connection Reuse (*ConnReuse*)

CDNs typically employ *ConnReuse*, allowing a new request to reuse older TCP connection. The key advantage of this feature is that the new request inherits matured congestion window (*cwnd*) and does not restart the connection from scratch, i.e. ICW. To analyze connection reuse, we analyzed the trace (GlobalCDN) to identify if and when requests reused existing connections and modified our setup to employ the reused connection's *cwnd* as the ICW for the page load[10].

Figure 8 plots Configanator improvement over *ConnReuse*. We observe that Configanator gains are reduced from 18% over *Default* (Figure 5b) to 14% at the median. The benefits at the tail are still substantial, with 56% p95 improvement. There are several reasons for this behavior: First, connection reuse only impacts the slow-start phase (e.g., ICW) and does not tune the CCA and HTTP, the top two critical knobs (Figure 13). Second, even with reuse, a connection is not always guaranteed to reuse the old *cwnd*, since other TCP settings like *slow_start_after_idle* may reset to default ICW — forcing a reused connection to again go through slow start phase. In fact, the old *cwnd* is reused with a probability of 0.27 in our trace, i.e., only a small subset of requests exploit the benefits of reuse. Third, unlike Configanator's exploitation of good configurations for similar connections, the scope of *ConnReuse* if limited to a single connection — a new connection from even the same user will go through the default slow start phase. Consequently, while connection reuse outperforms the *Default* by only 4.65% and 19.6% at median and tail respectively, a variant

[10]We infer *ConnReuse* if the first *cwnd* for a request is greater than connection's ICW. Our GlobalCDN trace directly captures these fields for each request. Note that, the reused connection may also inherit the MTU and SRTT values, but we limit our focus to the key component that limit data transfer, i.e., *cwnd*.

NC group	PLT ratio	Prefix distribution				
		p5	p25	p50	p75	95
<=5	Global / Configanator	0.92	1.0	1.0	1.0	1.08
	Default/Configanator	1.05	1.07	1.14	1.27	2.34
>=30	Global / Configanator	0.89	0.96	1.0	1.0	1.06
	Default/Configanator	1.04	1.09	1.17	1.21	2.67

Table 3: Impact of NC size on performance.

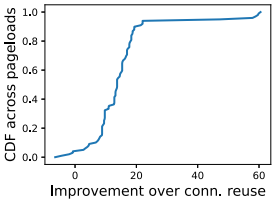


Figure 8: Impact vs TCP connection reuse.

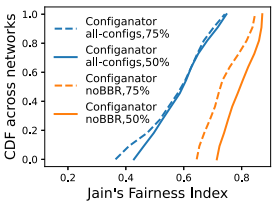


Figure 9: Impact on fairness.

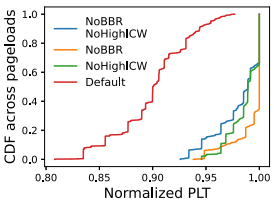


Figure 10: Only fair configurations.

of Configanator that only tunes *ICW* still performs *ConnReuse* with 8.7% and 23.4% improvements at median and tail.

These results suggest that *ConnReuse* alone is not the silver bullet, also portrayed by other ICW tuning system [42], and Configanator is expected to bring substantial improvements even when traditional optimizations are considered.

6.6 System Benchmarks

Next, we evaluate the latency, CPU and memory overheads. The experiments are performed in a testbed by emulating network conditions from our traces for 10 randomly selected websites. We repeat each test 1000 times.

Latency Overheads: For latency overheads, we focus on the modifications to ALPN to enable HTTP level tuning. We compare Configanator against a version that does not modify ALPN and tunes HTTP level by renegotiates which incurs at least 1-RTT overhead. Figure 11 plots the PLT for the two variants, normalized by *Default* (vanilla Apache). Given that Configanator simply edits the "ALPN next protocol" field in TLS *Server_Hello* without requiring any extra communication, we observe no latency overheads and a similar performance to the *Default*. For *Renegotiation*, we observe a slight PLT inflation ($\sim$3% at the median) which is due to the TLS renegotiation required to switch the HTTP version. We note that this approach still has a minor overhead (4% higher PLT at median) because a page load requires many RTTs and this overhead get amortized.

CPU and Memory Overheads To measure the CPU and memory overheads, we leveraged the Apache Benchmark tool to setup 100, 250 and 500 concurrent connections. We observe slight CPU overhead ($<$5%) as compared to *Default*. Although reconfiguring the connections do not require any additional memory, keeping the IP prefixes and their NC/configuration rules in the KV-store contributed to an increased memory usage.

6.7 Fairness Implications

Next, we explore the fairness implications. Within the testbed, we explore the situation where 30 concurrent flows share a representative bottleneck link, i.e., the access links for 3G, 4G, etc (number of flows from [115] Appendix F.10), under shallow buffers ({0.5 and 1} BDP). We use *Jain's Fairness Index* [63] to quantify fairness. We split the connections into two groups – one using Configanator and another using the default configuration (e.g., Cubic with 10MSS ICW). We then vary the percentage of connections in each group.

Quantifying Unfairness: Figure 9(a) present Jain's index when 75% and 50% of the flows are tuned. We observe that fairness decreases as the percentage of Configanator-tuned flows increases. Unsurprisingly, unfairness arises for two reasons: (1) when a flow is configured to use BBR [24,57,111,129,137], and (2) when a flow is configured to use high ICW values (even if BBR is not used) [52,72,88].

Configanator without unfair configurations: Next, we excluded the unfair configurations from the configuration space and tested 3 scenarios: (i) prevent BBR usage, (ii) prevent high ICW usage, (iii) prevent both. Figure 10 plots the ratio of PLT seen for vanilla Configanator (all configuration) to the variants, for GlobalCDN traces. We observe that NoBBR and NoHighICW perform similar to vanilla system for a significant fraction of the trace (63% and 35% respectively) and within 6% for worst case: this is because BBR is not always the optimal choice and application layer tuning (HTTP version) helps account for the lack of BBR or HighICW.

The results show that Configanator can provide an alternate war-chest to CDNs to improve web performance, even without using the unfair configurations.

6.8 Critical Knobs

We analyze the relative importance of reconfiguring different configuration parameters (Table 1). Our goal is to understand the minimal (or critical) parameters that must be tuned to significantly improve performance. In Figure 13, we plot the performance benefits of using distinct subset of configuration parameters, leveraging the brute-force exploration data from PLT-Tensor. We observe that the top 3 crucial parameters are HTTP version, congestion control algorithm (TCP-CC) and ICW. Moreover, when performing a layer to layer comparison, we observe that the Transport layer parameters combined (*Tran. layer*) have a higher impact on performance than the Application layer knobs combined(*App layer*). To explain this discrepancy, we analyze the different knobs in each layer and we observed that while certain transport knobs, e.g., Auto Corking, have little benefit in the median scenario, they are influential at the tails. Unlike the transport layer, in the Application layer most of the parameters (e.g., HTTP2 settings like header table size etc.) do not show significant benefit in median or tail conditions.

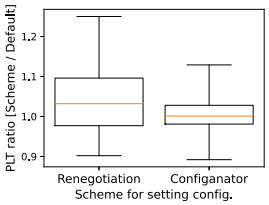


Figure 11: Latency overheads.

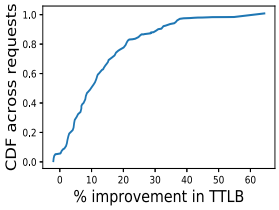


Figure 12: Reconfiguration benefits for CDN traffic

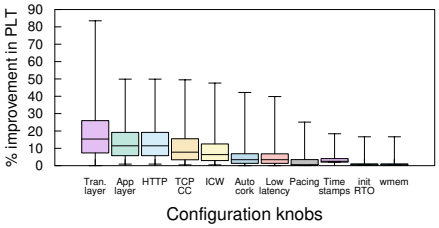
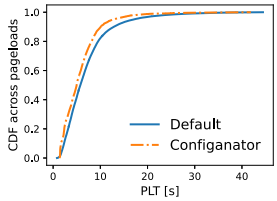


Figure 13: Critical configuration parameters



Pageload	PLT diff. (ms)	% imp.
p25	671	13.7
p50	767	14.6
p75	1219	21.5
p95	3797	26.3

Figure 14: Live deployment PLTs and improvements.

7 Live Deployment

In this section, we present the results for dynamically tuning the configurations at scale through a controlled experiment at GlobalCDN and a live prototype deployment on Google Cloud with 3161 end-users.

7.1 Validation at GlobalCDN

Next, we validate our approach when applied to data with more realistic and diverse client settings. We conducted measurements at GlobalCDN to collect data regarding performance of different configurations for the diverse networks and end-user devices. Specifically, we used the *default* configuration for 80% of the connections and explored random configurations for the rest to generate the data needed to emulate the contextual multi-armed bandit based exploration. Due to operational constraints, we analyzed a subset of configuration knobs: different congestion control algorithms and ICW. The experiments were conducted by randomly selecting 1% of the users from 3 of the CDN PoPs, for a duration of 6 hours. Note, these PoPs have the same workloads as the GlobalCDN trace described earlier.

We replayed the captured traces in our simulator, with two key distinctions: (i) the testbed-based PLT-Tensor was replaced by TTLB measurements collected from production users, since these TTLB measurements encompass the performance across real-world users, (ii) This experiment covers diverse user devices in-the-wild. Figure 12 presents the TTLB improvements for Configanator (versus default configuration) with upto 37% improvement at the tail (p95). Although this simulation covers a smaller configuration space, the improvements affirm the efficacy of tuning at scale, working with the diverse set of NC features (user device, network, geo-location, AS).

7.2 Google Cloud Deployment

We deployed Configanator on several Google Cloud servers, each with 8 CPU cores and 32 GB of RAM. We evenly divide the servers into two groups: one half with the Configanator-enhanced servers, while the other half with traditional Apache server. We cloned a variety of real-world websites from Alexa top-100 and hosted them on servers without sharding. We hosted the Configuration Manager on a dedicated instance.

For clients, we used SpeedChecker [81, 82], a platform for global Internet measurements with vantage points deployed across the globe. We had 3161 clients in total, spread across 4 of the continents. The clients periodically conducted pageloads from both the Configanator and the traditional web servers at the same frequency, resulting in ∼150K pageloads in 21 days. Further details about SpeedChecker are provided in Appendix F.1.

Figure 14 plots the raw PLTs observed for the two systems, with the accompanying table summarizing the PLT difference and improvements. Due to the online nature of the exploration and learning, we observe PLT degradation for a small subset of pageloads: 4.3% of the pageloads faced upto -13% degradation. For the rest, Configanator resulted in significant improvements, with upto 3.8s improved PLT at the tail (upto 767ms for the median). Dissecting the improvements across networks and websites, we observe a trend similar to Figure 5c: low bandwidth, high RTT/loss networks and content-rich websites get the most benefits. For the top configurations, we observe no clear winner: top 5 covered 3 CCs (BRR, Cubic and Vegas), both HTTP versions, and ICW randing from 16 to 40. We observe a stark difference in the ICW values used by clients in developed regions (Europe, N.America), with higher ICW (30-50 MSS), compared to developing regions (16-24 MSS).

Most of the clients (∼75%) are from N.America/Europe and the rest are geographically distributed which results in unbalanced Network Classes (NCs), leading to a higher share of traffic for the probes in N.America/Europe. Interestingly, NCs with the most number of pageloads, although showing good improvements (11-13% at median), are not the ones where we observe the highest benefits, owing to their good bandwidth, low RTT connections. We observe that the less-dense NCs still outperform Default (by more than 8% at median), since Configanator's exploitation arm is able to generalize to a modest extent by using data collected across all NCs.

8 Discussion and Limitations

Security and Equilibrium: Potential implications of self-learning systems include adversarial attacks [123] or oscillations. We are working to formulate the interactions

between different instances of Configanator (i.e., deployments by different CDNs) as a game-theoretic problem to understand our system at equilibrium.

Management Overheads: Dynamically reconfiguring the CDN's protocol stack complicates performance diagnosis. We plan to investigate methods for reducing this complexity, e.g., minimizing the number of active configuration combinations. Further, different configurations may vary in their resource-consumption at the CDN edge and we plan to investigate the configuration associated resource-overheads in the future.

Data Bias: Configanator's data-driven workflow can be impacted by the inherent biases of trace-driven systems [11], e.g., choice of configuration can have an impact on the feedback loop's decision features. We leave a more comprehensive analysis of biasness to future work.

Testbed Limitations: Owing the lack of cellular connections and devices in the testbed, our simulator is unable to emulate different end-user devices and cellular last-miles. Although the dataset from GlobalCDN covers diverse last-mile connections and devices, we plan to explore systematic approaches to incorporate this diversity in the testbed.

Trace Limitations: While several of our traces capture end-to-end behavior (GlobalCDN, Pantheon, FCC), two of our traces do not. Specifically, CAIDA and MAWI traces are from core router and we recreate end-to-end behavior by matching data with ACKs: this recreation can introduce some imprecision into our latency, loss and BW calculations.

NC Size Bias: As demonstrated in the evaluation, connection homogeneity within an NC (due to small NC size) favors Configanator. This bias is prevalent in two of our traces, FCC and Pantheon (comprising synthetically generated flows). However, this does not hold for the realistic traces (e.g., GlobalCDN) which are mainly focused in the evaluation when discussing the size bias, and still shows improvements over the Default.

9 Related Work

Web Performance Many measurement studies [3, 33, 36, 48, 135] have explored the performance of different networking protocol settings and the impact of tuning on web performance. Our system builds on the observations from these studies: namely that different configurations are required for different network conditions and websites. Web improvement by cross-layer tuning was earlier motivated in [91] (Configanator's workshop paper) and the present paper builds a practical algorithm and system for tuning the configurations. It further evaluates idea in a wide range of realistic scenarios.

Self-Tuning Systems: Self-tuning systems have been explored within the context of transport protocols [31, 64, 77, 98–100, 113, 138], video [2, 68, 85, 124], databases [32, 56, 131], and cloud systems [4, 13, 78, 147]. While our work shares a similar ideology of exploiting heterogeneity, we differ in our methods for learning optimal configuration and in the domain specific solution for implementing reconfiguration. While [68, 85] employ similar multi-armed bandits, our bandit generalizes across clusters and includes a Gaussian process to speed up learning. Additionally, while some model relatively static or offline workloads [2, 4, 32, 131], Configanator takes an online approach to tackle network and workload dynamics. Unlike [138] which rely on priori assumptions of the network, Configanator builds a performance model-based on live feedback which allows it to adapt to network dynamics. In contrast with [31, 64, 77, 79, 98–100, 113] which focus on tuning specific aspects of stack, Configanator tunes across a broader set of layers and parameters. Similarly, while these techniques use features from only network, Configanator also incorporates application features (e.g., website).

While Configanator focuses on control over server configurations, others [48, 109] require control over both the servers and the network switches to perform appropriate learning and tuning — applicable to data centers. Others [7, 90] move CCA outside data-path, enabling fast development and portability. Such innovative techniques simplify the design of Configanator by externalizing and simplifying tuning.

CrossLayer Optimizations. We differ from existing cross-layer optimizations [3, 9, 15, 25] which introduce APIs to enable the different layers to communicate and react accordingly the network events. Instead, we externalize the optimization logic and present an interface across the different layers to enable an external entity to configure the different layers which requires a learning algorithm agnostic of applications – a key contribution of Configanator.

10 Conclusion

In this paper, we argue that "one-size-fits-all" approach to configuring web server's network stack results in sub-par performance for end-users, especially those in emerging regions. Due to the ever-expanding nature of Internet, all end-users do not face similar network conditions. This argument stands in stark contrast to the traditional setup of today's web serving stacks where a single configuration is used for a divergent set of users.

This paper takes the first step towards realizing heterogeneity and fine-grained reconfiguration in a principled and systematic manner: our system, Configanator, introduces a principled framework for learning better configurations, than the default, for a connection by systematically exploring the performance of different configurations across a set of similar connections. We demonstrate the benefits of Configanator using both a live deployment and a large scale simulation.

11 Acknowledgments

We thank the anonymous reviewers, Michael Schapira (our shepherd), Zachary Bischof, Luca Niccolini, Ranjeeth Dasineni, Huapeng Zhou and Ali Razeen for their invaluable feedback on earlier drafts of this paper. We also thank Janusz Jezowicz from SpeedChecker for granting us access to their platform. This work is supported by NSF grants CNS-1819109, CNS-1814285 and Richard B. Salomon Faculty Research Award.

References

[1] AHMAD, S., HAAMID, A. L., QAZI, Z. A., ZHOU, Z., BENSON, T., AND QAZI, I. A. A view from the other side: Understanding mobile phone characteristics in the developing world. In *Proceedings of the 2016 ACM on Internet Measurement Conference* (2016), ACM, pp. 319–325.

[2] 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* (2018), ACM, pp. 44–58.

[3] AL-FARES, M., ELMELEEGY, K., REED, B., AND GASHINSKY, I. Overclocking the yahoo!: Cdn for faster web page loads. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (2011), ACM, pp. 569–584.

[4] ALIPOURFARD, O., LIU, H. H., CHEN, J., VENKATARAMAN, S., YU, M., AND ZHANG, M. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI* (2017), pp. 469–482.

[5] ALTHOUSE, J. Tls fingerprinting with ja3 and ja3s. https://sforce.co/3kUXKv8.

[6] ANDERSON, B. Tls fingerprinting in the real world. https://bit.ly/3l3Jnoe.

[7] ARASHLOO, M. T., GHOBADI, M., REXFORD, J., AND WALKER, D. Hotcocoa: Hardware congestion control abstractions. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (2017), ACM, pp. 108–114.

[8] ARCHIVE, M. W. G. T. Packet traces from wide backbone 12/1/17 to 12/7/17. http://mawi.wide.ad.jp/mawi/.

[9] BALAKRISHNAN, H., RAHUL, H. S., AND SESHAN, S. An integrated congestion management architecture for internet hosts. *ACM SIGCOMM Computer Communication Review 29*, 4 (1999), 175–187.

[10] BALAKRISHNAN, H., STEMM, M., SESHAN, S., AND KATZ, R. H. Analyzing stability in wide-area network performance. *ACM SIGMETRICS Performance Evaluation Review 25*, 1 (1997), 2–12.

[11] BARTULOVIC, M., JIANG, J., BALAKRISHNAN, S., SEKAR, V., AND SINOPOLI, B. Biases in data-driven networking, and what to do about them. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (2017), ACM, pp. 192–198.

[12] BELSHE, M., PEON, R., AND THOMSON, M. Hypertext transfer protocol version 2 (http/2). https://http2.github.io/http2-spec/.

[13] BILAL, M., AND CANINI, M. Towards automatic parameter tuning of stream processing systems. In *Proceedings of the 2017 Symposium on Cloud Computing* (New York, NY, USA, 2017), SoCC '17, ACM, pp. 189–200.

[14] BOJAN PAVIC, CHRIS ANSTEY, J. W. Why does speed matter? https://web.dev/why-speed-matters/.

[15] BRIDGES, P. G., WONG, G. T., HILTUNEN, M., SCHLICHTING, R. D., AND BARRICK, M. J. A configurable and extensible transport protocol. *IEEE/ACM Transactions on Networking 15*, 6 (2007), 1254–1265.

[16] BROCHU, E., CORA, V. M., AND DE FREITAS, N. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599* (2010).

[17] BROIDO, A., AND CLAFFY, K. Analysis of RouteViews BGP data: policy atoms. In *Network Resource Data Management Workshop* (Santa Barbara, CA, May 2001).

[18] BROTHERSTON, L. Fingerprintls. https://bit.ly/3BJfFuK.

[19] BRUTLAG, J. Speed matters for google web search, 2009.

[20] BUTKIEWICZ, M., MADHYASTHA, H. V., AND SEKAR, V. Understanding website complexity: measurements, metrics, and implications. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (2011), ACM, pp. 313–328.

[21] BUTKIEWICZ, M., WANG, D., WU, Z., MADHYASTHA, H. V., AND SEKAR, V. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *NSDI* (2015), vol. 1, pp. 2–3.

[22] CAIDA. The caida ucsd anonymized internet traces 2016 dataset. https://bit.ly/3l1AVG6.

[23] CARDWELL, N., CHENG, Y., GUNN, C. S., YEGANEH, S. H., ET AL. Bbr: congestion-based congestion control. *Communications of the ACM 60*, 2 (2017), 58–66.

[24] CARDWELL, N., CHENG, Y., YEGANEH, S. H., SWETT, I., VASILIEV, V., JHA, P., SEUNG, Y., MATHIS, M., AND JACOBSON, V. Bbr v2 a model-based congestion control. https://bit.ly/3x4bQwx.

[25] CHEN, A., SRIRAMAN, A., VAIDYA, T., ZHANG, Y., HAEBERLEN, A., LOO, B. T., PHAN, L. T. X., SHERR, M., SHIELDS, C., AND ZHOU, W. Dispersing asymmetric ddos attacks with splitstack. In *HotNets* (2016), pp. 197–203.

[26] CHEN, F., SITARAMAN, R. K., AND TORRES, M. End-user mapping: Next generation request routing for content delivery. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 167–181.

[27] CUI, Y., DAI, N., LAI, Z., LI, M., LI, Z., HU, Y., REN, K., AND CHEN, Y. Tailcutter: Wisely cutting tail latency in cloud cdns under cost constraints. *IEEE/ACM Transactions on Networking 27*, 4 (2019), 1612–1628.

[28] DEAN, J., AND BARROSO, L. A. The tail at scale. *Communications of the ACM 56* (2013), 74–80.

[29] DEV@TRAFFICSERVER.APACHE.ORG. Ja3 fingerprint plugin. https://bit.ly/3iTg70U.

[30] DIGITAL, D. Milliseconds make millions: A study on how improvements in mobile site speed positively affect a brand's bottom line. https://bit.ly/3rpm8WP.

[31] DONG, M., LI, Q., ZARCHY, D., GODFREY, P. B., AND SCHAPIRA, M. Pcc: Re-architecting congestion control for consistent high performance. In *NSDI* (2015), vol. 1, p. 2.

[32] DUAN, S., THUMMALA, V., AND BABU, S. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment 2*, 1 (2009), 1246–1257.

[33] DUKKIPATI, N., REFICE, T., CHENG, Y., CHU, J., HERBERT, T., AGARWAL, A., JAIN, A., AND SUTIN, N. An argument for increasing tcp's initial congestion window. *Computer Communication Review 40*, 3 (2010), 26–33.

[34] DUMAZET, E. tcp: provide syn headers for passive connections. https://lwn.net/Articles/645128/.

[35] DUNKELS, A., ET AL. The lwip tcp/ip stack. *lwIP–A LightWeight TCP/IP Stack* (2004).

[36] ERMAN, J., GOPALAKRISHNAN, V., JANA, R., AND RAMAKRISHNAN, K. K. Towards a spdyier mobile web? *IEEE/ACM Transactions on Networking 23*, 6 (2015), 2010–2023.

[37] FACEBOOK. Aquila (internet deployment by drone). https://bit.ly/2V92Adk.

[38] FACEBOOK. Network connection class. https://github.com/facebook/network-connection-class.

[39] FACEBOOK. Proxygen: Facebook's c++ http libraries. https://github.com/facebook/proxygen.

[40] FASTLY. Advanced tcp optimizations. https://bit.ly/3f2SstA.

[41] FCC. Measuring fixed broadband report - 2016. https://bit.ly/2TAxef8.

[42] FLORES, M., KHAKPOUR, A. R., AND BEDI, H. Riptide: Jump-starting back-office connections in cloud systems. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)* (2016), IEEE, pp. 78–87.

[43] FOUNDATION, L. *Open vSwitch*.

[44] FOUNDATION, O. S. Openssl alpn callback. https://bit.ly/3rG5rXh.

[45] FOWLER, D. Cdn tuning for ott - why doesn't it already do that? https://bit.ly/3i4z7Kr.

[46] GANJAM, A., SIDDIQUI, F., ZHAN, J., LIU, X., STOICA, I., JIANG, J., SEKAR, V., AND ZHANG, H. C3: Internet-scale control plane for video quality optimization. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)* (2015), pp. 131–144.

[47] GARDNER, J. R., KUSNER, M. J., XU, Z. E., WEINBERGER, K. Q., AND CUNNINGHAM, J. P. Bayesian optimization with inequality constraints. In *ICML* (2014), pp. 937–945.

[48] GHOBADI, M., YEGANEH, S. H., AND GANJALI, Y. Rethinking end-to-end congestion control in software-defined networks. In *Proceedings of the 11th ACM Workshop on Hot Topics in networks* (2012), ACM, pp. 61–66.

[49] GONG, S., NASEER, U., AND BENSON, T. Inspector gadget: A framework for inferring tcp congestion control algorithms and protocol configurations. In *Network Traffic Measurement and Analysis Conference (TMA)* (2020).

[50] GOOGLE. The chromium projects. https://www.chromium.org/.

[51] GOOGLE. Quic, a multiplexed stream transport over udp. https://www.chromium.org/quic.

[52] GRIECO, L. A., AND MASCOLO, S. Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control. *ACM SIGCOMM Computer Communication Review 34*, 2 (2004), 25–38.

[53] HELT, J., FENG, G., SESHAN, S., AND SEKAR, V. Sandpaper: mitigating performance interference in cdn edge proxies. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing* (2019), pp. 30–46.

[54] HEMMINGER, S. Netem - network emulator. https://bit.ly/36ZXT8k.

[55] HERBERT, T. tcp: Socket option to set congestion window. https://bit.ly/3zDnH6r.

[56] HERODOTOU, H., LIM, H., LUO, G., BORISOV, N., DONG, L., CETIN, F. B., AND BABU, S. Starfish: A self-tuning system for big data analytics. In *Cidr* (2011), vol. 11, pp. 261–272.

[57] HOCK, M., BLESS, R., AND ZITTERBART, M. Experimental evaluation of bbr congestion control. In *Network Protocols (ICNP), 2017 IEEE 25th International Conference on* (2017), IEEE, pp. 1–10.

[58] HOFF, T. Latency is everywhere and it costs you sales - how to crush it. https://bit.ly/3x5u3Kb.

[59] IETF. Rfc 6298. https://tools.ietf.org/html/rfc6298.

[60] IETF. Rfc 7301 transport layer security (tls) application-layer protocol negotiation extension. https://tools.ietf.org/rfc/rfc7301.txt.

[61] INC., C. Malcolm measuring active listeners, connection observers, and legitimate monitors. https://malcolm.cloudflare.com/.

[62] INC., M. Geoip2 city database. https://www.maxmind.com/en/geoip2-city.

[63] JAIN, R., DURRESI, A., AND BABIC, G. Throughput fairness index: An explanation. In *ATM Forum contribution* (1999), vol. 99.

[64] JAY, N., ROTMAN, N. H., GODFREY, P., SCHAPIRA, M., AND TAMAR, A. Internet congestion control via deep reinforcement learning. *arXiv preprint arXiv:1810.03259* (2018).

[65] JEONG, E., WOO, S., JAMSHED, M. A., JEONG, H., IHM, S., HAN, D., AND PARK, K. mtcp: a highly scalable user-level tcp stack for multicore systems. In *NSDI* (2014), pp. 489–502.

[66] JIANG, J., DAS, R., ANANTHANARAYANAN, G., CHOU, P. A., PADMANABHAN, V., SEKAR, V., DOMINIQUE, E., GOLISZEWSKI, M., KUKOLECA, D., VAFIN, R., ET AL. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016), ACM, pp. 286–299.

[67] JIANG, J., SEKAR, V., MILNER, H., SHEPHERD, D., STOICA, I., AND ZHANG, H. Cfa: A practical prediction system for video qoe optimization. In *NSDI* (2016), pp. 137–150.

[68] JIANG, J., SUN, S., SEKAR, V., AND ZHANG, H. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), USENIX Association, pp. 393–406.

[69] JIN, Y., RENGANATHAN, S., ANANTHANARAYANAN, G., JIANG, J., PADMANABHAN, V. N., SCHRODER, M., CALDER, M., AND KRISHNAMURTHY, A. Zooming in on wide-area latencies to a global cloud provider. In *Proceedings of the ACM Special Interest Group on Data Communication* (2019), ACM, pp. 104–116.

[70] JOHN B. ALTHOUSE, JEFF ATKINSON, J. A. Ja3 - a method for profiling ssl/tls clients. https://github.com/salesforce/ja3.

[71] KAYSER, B. What is the expected distribution of website response times?

[72] KOZU, T., AKIYAMA, Y., AND YAMAGUCHI, S. Improving rtt fairness on cubic tcp. In *2013 First International Symposium on Computing and Networking* (2013), IEEE, pp. 162–167.

[73] KRAUSE, A., AND ONG, C. S. Contextual gaussian process bandit optimization. In *Advances in neural information processing systems* (2011), pp. 2447–2455.

[74] LEE, Y., AND SPRING, N. Identifying and aggregating homogeneous ipv4 /24 blocks with hobbit. In *Proceedings of the 2016 Internet Measurement Conference* (New York, NY, USA, 2016), IMC '16, ACM, pp. 151–165.

[75] LETHAM, B., KARRER, B., OTTONI, G., AND BAKSHY, E. *Efficient tuning of online systems using Bayesian optimization*. https://bit.ly/3rBMHIm.

[76] LI, L., CHU, W., LANGFORD, J., AND SCHAPIRE, R. E. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web* (2010), ACM, pp. 661–670.

[77] LI, W., ZHOU, F., CHOWDHURY, K. R., AND MELEIS, W. M. Qtcp: Adaptive congestion control with reinforcement learning. *IEEE Transactions on Network Science and Engineering* (2018).

[78] LI, Z. L., LIANG, M. C.-J., HE, W., ZHU, L., DAI, W., JIANG, J., AND SUN, G. Metis: Robustly tuning tail

latencies of cloud systems. In *ATC (USENIX Annual Technical Conference)* (July 2018), USENIX.

[79] LIU, H. H., VISWANATHAN, R., CALDER, M., AKELLA, A., MAHAJAN, R., PADHYE, J., AND ZHANG, M. Efficiently delivering online services over integrated infrastructure. In *NSDI* (2016), vol. 1, p. 1.

[80] LOON, P. Balloon powered internet. https://x.company/loon/.

[81] LTD., S. Speedchecker. https://probeapi.speedchecker.com/.

[82] LTD., S. Speedchecker - probe api documentation. https://bit.ly/2TGwdCu.

[83] 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.

[84] LU, T., PÁL, D., AND PÁL, M. Contextual multi-armed bandits. In *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics* (2010), pp. 485–492.

[85] MAO, H., NETRAVALI, R., AND ALIZADEH, M. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), ACM, pp. 197–210.

[86] MCKAY, M. D., BECKMAN, R. J., AND CONOVER, W. J. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics 21*, 2 (1979), 239–245.

[87] MCQUISTIN, S., UPPU, S. P., AND FLORES, M. Taming anycast in the wild internet. In *Proceedings of the Internet Measurement Conference* (2019), pp. 165–178.

[88] MO, J., LA, R. J., ANANTHARAM, V., AND WALRAND, J. Analysis and comparison of tcp reno and vegas. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)* (1999), vol. 3, IEEE, pp. 1556–1563.

[89] MUKERJEE, M. K., NAYLOR, D., JIANG, J., HAN, D., SESHAN, S., AND ZHANG, H. Practical, real-time centralized control for cdn-based live video delivery. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 311–324.

[90] NARAYAN, A., CANGIALOSI, F., RAGHAVAN, D., GOYAL, P., NARAYANA, S., MITTAL, R., ALIZADEH, M., AND BALAKRISHNAN, H. Restructuring endpoint congestion control. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (2018), ACM, pp. 30–43.

[91] NASEER, U., AND BENSON, T. Configtron: Tackling network diversity with heterogeneous configurations. In *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)* (Santa Clara, CA, July 2017), USENIX Association.

[92] NASEER, U., AND BENSON, T. Inspectorgadget: Inferring network protocol configuration for web services. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (July 2018), pp. 1624–1629.

[93] NASEER, U., BENSON, T. A., AND NETRAVALI, R. Webmedic: Disentangling the memory-functionality tension for the next billion mobile web users. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (2021), pp. 71–77.

[94] NEJATI, J., AND BALASUBRAMANIAN, A. An in-depth study of mobile browser performance. In *Proceedings of the 25th International Conference on World Wide Web* (2016), International World Wide Web Conferences Steering Committee, pp. 1305–1315.

[95] NETRAVALI, R., GOYAL, A., MICKENS, J., AND BALAKRISHNAN, H. Polaris: Faster page loads using fine-grained dependency tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (2016), USENIX Association.

[96] NETRAVALI, R., SIVARAMAN, A., DAS, S., GOYAL, A., WINSTEIN, K., MICKENS, J., AND BALAKRISHNAN, H. Mahimahi: Accurate record-and-replay for http. In *USENIX Annual Technical Conference* (2015), pp. 417–429.

[97] NGINX. Nginx reverse proxy. https://bit.ly/3yapgbH.

[98] NIE, X., ZHAO, Y., CHEN, G., SUI, K., CHEN, Y., PEI, D., ZHANG, M., AND ZHANG, J. Tcp wise: One initial congestion window is not enough. In *Performance Computing and Communications Conference (IPCCC), 2017 IEEE 36th International* (2017), IEEE, pp. 1–8.

[99] NIE, X., ZHAO, Y., LI, Z., CHEN, G., SUI, K., ZHANG, J., YE, Z., AND PEI, D. Dynamic tcp initial windows and congestion control schemes through reinforcement learning. *IEEE Journal on Selected Areas in Communications 37*, 6 (2019), 1231–1247.

[100] NIE, X., ZHAO, Y., PEI, D., CHEN, G., SUI, K., AND ZHANG, J. Reducing web latency through dynamically setting tcp initial window with reinforcement learning.

[101] OF SHEFFIELD, M. L. G. U. Gpyopt. https://github.com/SheffieldML/GPyOpt.

[102] OF SHEFFIELD, M. L. G. U. Gpyopt.core.task.space module. https://bit.ly/3iVDuHf.

[103] OREILLY.COM. Bing and google agree: Slow pages lose users. https://bit.ly/374YGVI.

[104] PELIKAN, M., GOLDBERG, D. E., AND CANTÚ-PAZ, E. Boa: The bayesian optimization algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1* (1999), Morgan Kaufmann Publishers Inc., pp. 525–532.

[105] PI, Y., JAMIN, S., DANZIG, P., AND SHAHA, J. Apatoms: A high-accuracy data-driven client aggregation for global load balancing. *IEEE/ACM Transactions on Networking 26*, 6 (2018), 2748–2761.

[106] PICOTCP. picotcp. http://www.picotcp.com/.

[107] QUINLAN, J. R. Induction of decision trees. *Machine learning 1*, 1 (1986), 81–106.

[108] 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* (2017), ACM, pp. 390–403.

[109] RUFFY, F., PRZYSTUPA, M., AND BESCHASTNIKH, I. Iroko: A framework to prototype reinforcement learning for data center traffic control. *arXiv preprint arXiv:1812.09975* (2018).

[110] RÜTH, J., BORMANN, C., AND HOHLFELD, O. Large-scale scanning of tcp's initial window. In *Proceedings of the 2017 Internet Measurement Conference* (2017), ACM, pp. 304–310.

[111] RÜTH, J., KUNZE, I., AND HOHLFELD, O. An empirical view on content provider fairness. *arXiv preprint arXiv:1905.07152* (2019).

[112] RYZHOV, I. O. On the convergence rates of expected improvement methods. *Operations Research 64*, 6 (2016), 1515–1528.

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

[114] SCHOLZ, D., JAEGER, B., SCHWAIGHOFER, L., RAUMER, D., GEYER, F., AND CARLE, G. Towards a deeper understanding of tcp bbr congestion control. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops* (2018), IEEE, pp. 1–9.

[115] SCHULMAN, A., LEVIN, D., AND SPRING, N. CRAWDAD dataset umd/sigcomm2008 (v. 2009-03-02). Downloaded from https://crawdad.org/umd/sigcomm2008/20090302/pcap, Mar. 2009. traceset: pcap.

[116] SCIKIT LEARN.ORG. *Decision Trees*.

[117] SERVER, A. T. Tcpinfo plugin. https://bit.ly/3x8CyEr.

[118] SHAHRIARI, B., SWERSKY, K., WANG, Z., ADAMS, R. P., AND DE FREITAS, N. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE 104*, 1 (2015), 148–175.

[119] SHUFF, P. Building a billion user load balancer. USENIX Association.

[120] SINGH, S., MADHYASTHA, H. V., KRISHNAMURTHY, S. V., AND GOVINDAN, R. Flexiweb: Network-aware compaction for accelerating mobile web transfers. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (2015), ACM, pp. 604–616.

[121] SOFTWARE, V. Varnish http cache. https://varnish-cache.org/.

[122] STEIN, M. Large sample properties of simulations using latin hypercube sampling. *Technometrics 29*, 2 (1987), 143–151.

[123] SUN, Y., EDMUNDSON, A., VANBEVER, L., LI, O., REXFORD, J., CHIANG, M., AND MITTAL, P. Raptor: Routing attacks on privacy in tor.

[124] 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* (2016), ACM, pp. 272–285.

[125] SUNDARESAN, S., FEAMSTER, N., AND TEIXEIRA, R. Home network or access link? locating last-mile downstream throughput bottlenecks. In *International Conference on Passive and Active Network Measurement* (2016), Springer, pp. 111–123.

[126] TEAM, A. T. R. Bots tampering with tls to avoid detection. https://bit.ly/3f2cJjb.

[127] TLSFINGERPRINT.IO. Tls fingerprint. https://tlsfingerprint.io/.

[128] TRAN-THANH, L., CHAPMAN, A., DE COTE, E. M., ROGERS, A., AND JENNINGS, N. R. Epsilon–first policies for budget–limited multi-armed bandits. In *Twenty-Fourth AAAI Conference on Artificial Intelligence* (2010).

[129] TURKOVIC, B., KUIPERS, F. A., AND UHLIG, S. Interactions between congestion control algorithms. *network 3*, 17.

[130] URVOY-KELLER, G. On the stationarity of tcp bulk data transfers. In *International Workshop on Passive and Active Network Measurement* (2005), Springer, pp. 27–40.

[131] VAN AKEN, D., PAVLO, A., GORDON, G. J., AND ZHANG, B. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), ACM, pp. 1009–1024.

[132] VDMS. Our software - cdn. https://bit.ly/3iOMNbT.

[133] VERMOREL, J., AND MOHRI, M. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning* (2005), Springer, pp. 437–448.

[134] WANG, X. S., BALASUBRAMANIAN, A., KRISHNAMURTHY, A., AND WETHERALL, D. Demystifying page load performance with wprof. In *NSDI* (2013), pp. 473–485.

[135] WANG, X. S., BALASUBRAMANIAN, A., KRISHNAMURTHY, A., AND WETHERALL, D. How speedy is spdy? In *NSDI* (2014), pp. 387–399.

[136] WANG, X. S., KRISHNAMURTHY, A., AND WETHERALL, D. Speeding up web page loads with shandian. In *NSDI* (2016), pp. 109–122.

[137] WARE, R., MUKERJEE, M. K., SESHAN, S., AND SHERRY, J. Beyond jain's fairness index: Setting the bar for the deployment of congestion control algorithms. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (2019), pp. 17–24.

[138] WINSTEIN, K., AND BALAKRISHNAN, H. Tcp ex machina: computer-generated congestion control. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 123–134.

[139] WOHLFART, F., CHATZIS, N., DABANOGLU, C., CARLE, G., AND WILLINGER, W. Leveraging interconnections for performance: The serving infrastructure of a large cdn. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (2018), pp. 206–220.

[140] WONDRA, N. Magic transit: Network functions at cloudflare scale.

[141] YAN, F. Y., MA, J., HILL, G. D., RAGHAVAN, D., WAHBY, R. S., LEVIS, P., AND WINSTEIN, K. Pantheon datasets. https://pantheon.stanford.edu/measurements/node/.

[142] YAN, F. Y., MA, J., HILL, G. D., RAGHAVAN, D., WAHBY, R. S., LEVIS, P., AND WINSTEIN, K. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, 2018), USENIX Association, pp. 731–743.

[143] YANG, P., SHAO, J., LUO, W., XU, L., DEOGUN, J., AND LU, Y. Tcp congestion avoidance algorithm identification. *IEEE/ACM Transactions on Networking (TON) 22*, 4 (2014), 1311–1324.

[144] ZEROMQ. Zeromq. http://zeromq.org/.

[145] ZHANG, X., SEN, S., KURNIAWAN, D., GUNAWI, H., AND JIANG, J. E2e: embracing user heterogeneity to improve quality of experience on the web. In *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 289–302.

[146] ZHANG, Y., AND DUFFIELD, N. On the constancy of internet path properties. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement* (2001), ACM, pp. 197–211.

[147] ZHU, Y., LIU, J., GUO, M., BAO, Y., MA, W., LIU, Z., SONG, K., AND YANG, Y. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing* (2017), ACM, pp. 338–350.

Knob	Function
TargetAlgo	Sets corresponding tuning algorithm.
TargetNC	Controls the clustering strategy for NCs.
init_samples	Number of samples to initialize an NC.
NCSpread	Controls the performance spread that bounds a NC, and hence the number of cluster (K discussed in § 3.2).
AllowedConfig	Limits the space to disallow certain configurations.
PerfMemory	Length of history for configuration's performance over time.
UpdateLatency	Set latency b/w central *Config. Manager* and servers.
UpdateFreq	Controls the time after which a model is updated.
ChunkSize	Controls the time window for goodput, RTT and loss-rate measurements from packet traces.

Table 4: Simulator knobs

A Fingerprinting Configurations

Our fingerprinting techniques are inspired from recent works [49, 92, 110, 143]. Our tool inter-operates with TLS and infers configurations in the following ways: (i) HTTP configurations are visible to client during the connection setup and are fingerprinted from the server response, (ii) TCP configurations like RWIN are scraped from the packet headers, (iii) TCP initRTO is measured by emulating a loss during TCP handshake (i.e., by not acknowledging SYN packet back to the server), and measuring the time it takes the server to retransmit the SYN/ACKs, (iv) For TCP ICW, a big enough object URL is scraped from a website, the corresponding object is fetched and the number of packets sent by the server in first RTT is measured. Further, we use MSS=64B to trigger higher number of packets from server. We used AWS in respective regions as the vantage points for fingerprinting the configurations.

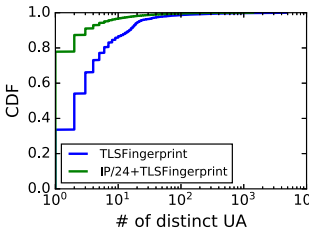


Figure 15: Relationship between TLS fingerprint and User-Agent.

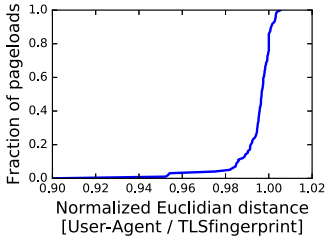


Figure 16: Comparison of device identifier's impact on NCs.

B TLS Fingerprinting for Device Identification

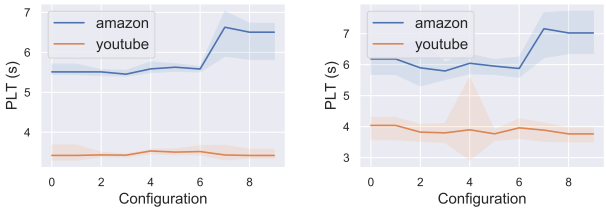
Recall that instead of the traditional User-Agent string, Configanator uses TLS fingerprinting for device identification as it allows device inference in early stages of the connection (prior to the HTTP version negotiation through ALPN). To evaluate its efficacy, we leverage a dataset from GlobalCDN, comprising 3.6M requests. The dataset consists of server logs and captures User-Agent strings from HTTP GET requests and the TLS fingerprint of the respective connections. The dataset includes 14.5K unique User-Agent strings and 3.2K unique TLS fingerprints.

Figure 15 plots the number of unique User-Agents (UA) that map to a TLS fingerprint. Ideally, a single UA should map to a fingerprint, thereby accurately identifying the corresponding device. However in practice, we observe that the one-to-one mapping is limited only to 34% of the fingerprints, with the rest mapping to atleast 2 UA. We observe that complementing the TLS fingerprint with the end-user IP-prefix helps in improving the accuracy, with 78% of the IP/24 and TLS fingerprint mapping to a single UA and 96% mapping to at-most 8 unique UA. We observe that for the cases where a single fingerprint maps to multiple UA strings, there are only minor differences, e.g., different browser versions, difference in OS's minor version (Android 6.0 vs 6.1.1).

In Figure 16, we further compare the two device identification techniques for clustering similar connections together. Using a dataset of 89K PLT measurements from GlobalCDN, we run our Network Class clustering using either User-Agent or TLS fingerprint as the basis for device identification. We compute the Euclidean distance of each connection PLT from its cluster's center and the figure plots the ratio of the distance. We observe that the ratio is between 0.98 and 1.00 for the overwhelming majority of the pageloads, indicating that the two technique perform fairly similar. Hence, device identification through TLS fingerprinting provides nearly similar accuracy to the User-Agent strings, with the added benefit that the device is identified prior to negotiating the HTTP version, whereas User-Agent string can only be inferred through HTTP requests headers (received after HTTP version negotiation).

C Passively Recording Network Conditions

Configanator passively collects goodput and packet loss rates for the IP-prefix (/24) and builds a historical archive (§ 3.2). When an IP connects, Configanator uses the handshake RTT and looks-up the goodput and packet loss rates from the recent session for the IP-prefix (/24) to aid in classifying the user into her Network Class. Configanator prototype uses Apache logs to collect information about user IP, the requested content, content size and download time. Additionally, per-connection TCP statistics are captured through Apache TCP Info plugin [117]. Using this information, Configanator calculates bandwidth (goodput) and packet loss rates on a per IP-prefix (/24) basis. Although we use heuristics for stable goodput and loss calculations, e.g., ignoring small objects, the goodput estimate may still under-estimate actual network bottleneck due to TCP mechanics (e.g., slow start phase). Consequently, the use of such measurements in the testbed (§ 6.1) may emulate lower bandwidths and higher loss rates (emulated loss plus induced buffer overflows) than the actual bottleneck links.



(a) Low spread (high K) (b) High spread (low K)

Figure 17: GP config-performance curve.

D Gaussian Process and Network Class Discussion

Bootstrapping GP: The first step of learning is to acquire data to bootstrap the Gaussian process. The bootstrap methodology is crucial for ensuring that the Gaussian-Bandit quickly finds good direction to explore. Recent works [4, 13, 32] have demonstrated the applicability of three distinct bootstrapping approaches: (i) *random*, in which the initial configurations are randomly selected; (ii) *domain-specific*, in which prior domain knowledge, captured through operator interviews or offline simulations, are used to rank configurations to sample; (iii) *Latin Hypercube Sampling* (LHS) which divides the input space into partitions and selects a sample from each partition to spread the samples evenly across space [122]. In this work, we use LHS to bootstrap the learning process. LHS has been found to aid bootstrapping Bayesian optimization by reaching an optimal decision quicker [86]. We observed LHS to speed up exploration in comparison with others by reducing the number of optimization steps by 2-3X, as the bootstrapping samples are spread evenly across space. A perfect rankings of configurations cannot be known prior to actually testing configurations, leading to ranking-based bootstrapping being sub-optimal to LHS.

Individual GP models for each website/Network Class: Bayesian Optimization is traditionally used for mapping configurations to their performance per workload (e.g., cloud configuration to cost [4]). Due to network dynamics and their implications on web performance [67, 135], a separate BO/GP model is required to map configuration performance for each workload (network condition and website), leading to individual exploration for each workload. The lack of cross network/website exploitation (due to separate BO models) makes a solely BO-based technique unfit for Configanator. Intuitively, the system should be able to generalize across networks and can use the already learnt pattern from other networks to a new network, e.g., HTTP/1.1 is optimal at high RTT, high loss for a complex website, no matter the bandwidth [135].

Figure 17 presents the GP model for two websites for the same set of configurations (x-axis) and the same Network Class (NC). In Figure 17a, while GP has estimated the curve for *youtube* with high confidence, *amazon* requires more data samples (wide confidence interval for configuration 7, 8 and 9). The different configuration-performance curves require a separate GP model for each website/NC for correct modeling of a configuration's performance and effective exploration, as the configuration-performance curve is distinct for every website and Network Class.

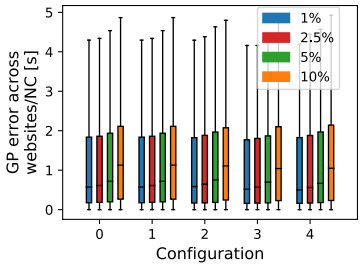


Figure 18: GP modeling error for different NC spread thresholds.

Impact of performance spread within NC: Next, in Figure 18, we leverage the *NCSpread* knob in simulator (Table 4) to test different bounds for Network Classes (NC) clustering. Recall that *NCSpread* controls the "K" for Kmeans clustering by selecting the lowest K that bounds the standard deviation of PLTs for a cluster's constituents within a specified threshold ({1, 2.5, 5, 10}% in the Figure 18). Determining the right K involves iterating through K values and is a three step process: (i) NC features – network characteristics (bandwidth, latency, loss rate), AS information (ASN, geo-location), and device type – from past connections are clustered using a given K, (ii) For each cluster, the list of PLTs observed for its members connections is generated and is normalized by the median PLT of the list[11], (iii) The standard deviation for each list is computed and, based on how far is it from the median and the *NCSpread* limit, the decision to converge on the given K or test a different K is made.

Figure 18 uses the testbed generated data from § 6 and plots the error in GP's estimate for five randomly selected configurations. The error is calculated as the absolute difference of GP's PLT estimate for a configuration and the actual PLT, and the boxes plot the error distribution observed for the various clusters (corresponding to the *NCSpread* value) and the websites. Note that, a small error is always expected due to the inherent variability with PLT measurements. The 5% limit *NCSpread* performs fairly close to the lower bounds, while also requiring a lower K: 7% lower K value than the 1% *NCSpread* threshold. This analysis serves as the motivation for using 5% value in the simulator.

Figures 17a and 17b further visualizes the confidence intervals for the GP models for 2 websites. For the high spread case (10% *NCSpread*), connections from slightly different

[11] As there might be multiple websites, there is one list per website. Further only PLTs for default configuration are used.

networks are mapped to the name GP, resulting in wider confidence intervals, and leads to inefficiency with acquisition function's next configuration suggestion.

E Deployment Considerations

Data-driven systems [2, 46, 66–68] traditionally use a split-plane architecture where a modeling layer (responsible for ingesting huge amounts of data and updating models) runs at a slower granularity than the decision layer (responsible for applying modeled decisions for users at real-time). Configanator's architecture uses a split-plane model which leverages the different computational requirements of Configanator's workflow: As demonstrated in Figure 4, in the slow path, the Configuration Manager collects telemetry from the web servers, uses this telemetry to update the learning model, and installs the configuration rules created by the model into the web servers. In the foreground, each web server uses the pre-installed rules to apply configuration to each connection and periodically collects telemetry from each connection.

The *first* phase, the background process, is time-consuming because of the process of updating the learning algorithms and Network Classes. The *second* phase, a fast, real-time process that applies the configuration rules to each inbound *user connection*, is run at the edge on each web server and provides low-latency, dynamic tuning. We note that although this decoupling results in the fast-path using stale information, we observe that this stale information still provides near-optimal performance [46].

F Supplementary Evaluation Material

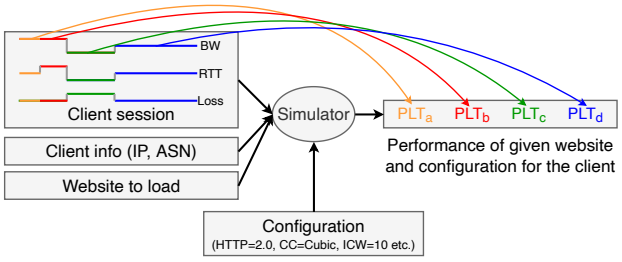


Figure 19: Simulating pageload for a client

F.1 Evaluation Setup

Simulation workflow: Figure 19 presents the workflow for simulating pageload performance. The client sessions are extracted from the real-world datasets and are modeled as time-series. Since we use 5s for measuring the network characteristics from the trace (a tunable knob as discussed in § 6.1), each linear state for BW, RTT, Loss in Figure 19 is at least 5s long. We extract an IP distribution from the trace to model the temporal aspects of client's connections (time at which a client connection (or IP) is seen in trace), i.e., the user sessions are fed to the simulator in the order they are observed in the real-world trace.

The simulator takes the goodput, RTT, loss rates at a certain time from the session time-series, client info (IP, ASN) and the target website to load as input. Using these features, it consults the configuration to test from the learning framework. Once the target configuration is known, it leverages the PLT-Tensor to map the network characteristics {goodput, RTT, loss-rate}, website and configuration to the eventual PLT. Note that, we assume that the network characteristics stay stable throughout the lifetime on a single pageload, supported by recent studies that TCP connection is piece-wise stationary and each segment stays stable in the order of tens of seconds to minutes [10].

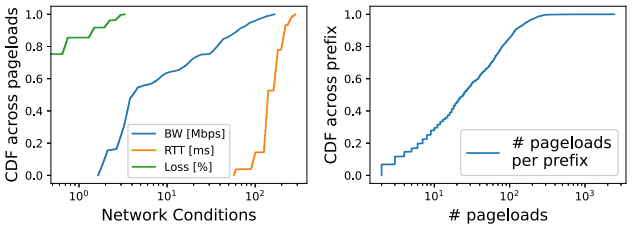
Table 4 further summarizes a number of simulator knobs that allow us to emulate and test a variety of scenarios.

Dataset description and breakdown: While the GlobalCDN, MAWI and CAIDA datasets are adequately described in § 6.1, here we provide details for the other two datasets.

The Pantheon dataset [141, 142] comprises of synthetically generated TCP flows across the different parts of the world. We collected three month's worth of data (May to July 2018) from Pantheon's website [141]. For the generated flows, the dataset logs the flow IDs, packet ingress/egress timestamps, packet sizes and one-way delay. Using these fields, we calculate the goodput, RTT and loss rates between each pair of end-point and, similar to the case for GlobalCDN, MAWI and CAIDA datasets, generate the time-series for the network characteristics. These end-points (vantage points) range from AWS deployments to university networks and cover multiple last-mile connection types. The FCC dataset is collected by the Measuring Broadband America program [41] and consists of a nation-wide study of end-user's broadband performance and an accompanying dataset. This dataset provides coarse granularity measurements in form of bandwidth, latency and loss rates distributions measured for real-world users. We use these distributions to generate synthetic traces, similar to [2].

The breakdown of ∼21.4M sessions is as follows: 8.2M from GlobalCDN, 2.7M from MAWI, 8.1M from CAIDA, 1.6M from FCC, 800K from Pantheon. The cross-regional nature of our datasets provide coverage over a wide range of representative network conditions, e.g., while FCC and CAIDA cover connections in U.S., MAWI dataset is from East Asia. Further, GlobalCDN and Pantheon [142] are even more diverse with connections from countries across the globe.

SpeedChecker and vantage points: SpeedChecker [81] is a platform for global Internet measurements, with vantage points deployed in over 170 countries and thousands of ISPs. SpeedChecker provides an API to conduct automated measurements ranging from ping, DNS, web pageloads to video tests. We leveraged vantage points (windows machines) on this platform for conducting the pageloads. The API call



(a) Network at vantage points (b) # of pageloads per prefix

Figure 20: Live-deployment vantage points

requires *CountryCode* and *Destinations* (a list of URLs to load). Vantage points (probes) from the specified country are selected internally by their platform and pageloads are conducted (upto 100 pageload every hour, per city in the country). Figures 20 presents various distributions about our vantage points. 20a compiles the distribution of network conditions observed for each pageload. The vantage points vary across the three dimensions and have mostly RTTs greater than 100ms. 20b presents the number of pageloads per prefix. We observe a heavy tail distribution, where certain vantage points conducted more pageloads than others, e.g., Europe, N.America had 4X more pageloads than Asia and Africa due to the higher number of the SpeedChecker clients in the developed regions. Africa had the smallest number of vantage points among all continents and the hourly limits were frequently reached, resulting in a lower number of total pageloads. Note that, diverse network conditions were still observed for the vantage points (Figure 20a) in spite of this skew in vantage point location. We further observe that 90% of the vantage points have unique IP-prefix (/24), showing that they are distributed and are not placed in a single facility, in the same subnet.

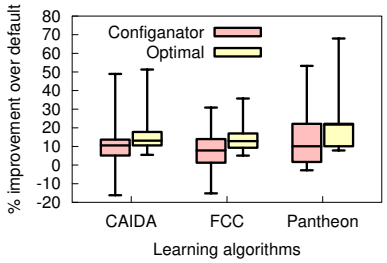


Figure 21: Configanator performance for CAIDA, FCC and Pantheon traces

F.2 Configanator Performance for CAIDA, FCC and Pantheon Traces

Figure 21 presents the distributions for Configanator's PLT improvement for the CAIDA, FCC and Pantheon traces. These figures complement the results in § 6.2 where we could not add the results for all the traces due to space limitations. CAIDA and FCC traces are collected from U.S.A and mostly cover high bandwidth, low RTT/loss connections, e.g., p95 RTT is 60ms. Following the trend observed in Figure 5c, we observe their PLT improvements over default to be lower as compared to other traces. Especially FCC dataset covers broadband connections and we observe the lowest p95 PLT improvement for FCC among all the datasets. Nevertheless, the improvements are still substantial with 610-640ms decrease in p95 PLT. On the other hand, Pantheon traces cover wider range of networks, often across continents, and result in upto 850ms improvement at tail.

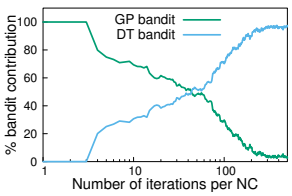


Figure 22: Bandits contribution

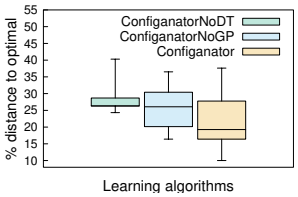


Figure 23: Tail performance

F.3 Bandit Contribution

Figure 22 uses the same convergence analysis as Figure 6 and plots the percentage of connections that uses a certain bandit. Initially GP bandit is largely used for a guided exploration. However, as more data is collected, DT bandit starts to overshadow the GP bandit, highlighting that a per-NC guided exploration is over-shadowed by cross-NC exploitation, when large data is available.

F.4 Bandit Performance at Tail

Figure 23 focuses on tail by dividing the entire trace into one minute segments and plotting the distance to optimal for the worst-case tail of each minute. Configanator's use of bandits enables it to perform better than individual bandits, being closer to optimal by more than 7%.

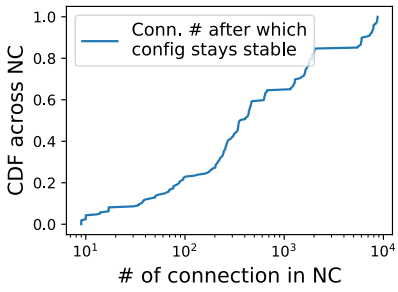


Figure 24: Time of last config. change in NC

F.5 Configuration Stability

Figure 24 plots the number of connections across NCs after which the DT-bandit's decision stays stable, i.e., configuration decision for the NC does not change. While for the median NC, the configuration choice becomes stable at ∼400th connection; we observe that it can take as much as 10K connections to reach the final configuration for some NCs. We observe the DT-bandit to stuck on a near-optimal configuration for these NCs. Down the line, the epsilon-bandit, randomly exploring, finds the optimal configuration and updates the NC. We note that Configanator switches to DT-bandit in the first 10-15 iterations for these NC, highlighting that the GP model's EI threshold was reached very early, and the initial exploration through GP was not very beneficial in uncovering the optimal configuration.

F.6 Design Choices for Network Classes

We use *GlobalCDN* dataset to evaluate design choices for classifying similar users together. We compare Configanator's clustering with: (i) *IP-Prefix* clusters /24 users together, (ii) *Hobbit* [74] improves /24 groups by merging dis-contiguous /24s based on co-location in Internet topology and homogeneous performance, (iii) *Latency Driven* inspired from AP-Atoms [105] where users with similar latency are grouped together, (iv) since CDNs group users based on their performance similarity [26, 119], *CDN Aggregation* use the natural CDN grouping and assigns all users mapped to a PoP to the same NC. We extract these mappings from the GlobalCDN dataset and, as these mappings can vary over time, build a time-series of user to CDN PoP mapping.

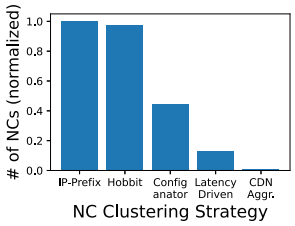


Figure 25: Impact of clustering on # of NC

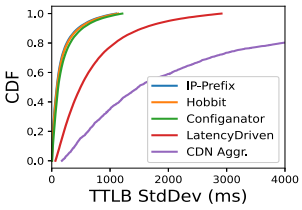
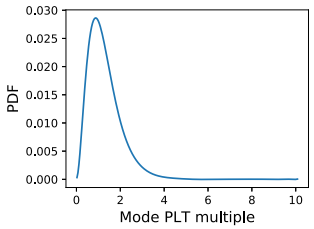
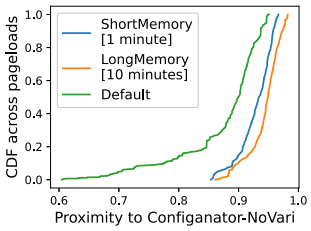


Figure 26: Spread of TTLBs within NC

Figure 25 and 26 plots the number of NCs and the spread of TTLBs within an NC for the different strategies. Ideally, Configanator favors small number of NCs and aims for small to negligible variations within NC performance metric, as the goal is to cluster similarly performing users together (§ 3.2). We observe *Hobbit* subnets /24 groups to have a poor coverage over the trace (Hobbit only covers 12% of prefixes in GlobalCDN dataset), with non-Hobbit /24s being treated as individual groups, leading to similar results as *IP-Prefix* (figure 25). Although NCs built by *Hobbit* and *IP-Prefix* have lowest performance divergence, (low std. dev. in figure 26); Configanator NCs are almost similarly compact, while using less than half



(a) Observed PLT variations.



(b) Impact of PLT variability.

Figure 27: PLT variability

number of NCs. Although *Latency Driven* uses least number of NCs, the lack of device, bandwidth, loss etc. information leads to diverse users being grouped together (high TTLB std. dev.). Similarly, since CDNs maps user based on latency to their closest PoPs, network and device heterogeneity still exist (e.g., the closest PoP to a user can be 10ms-600ms [26]), leading to highest performance variation within an NC for *CDN Aggr.*

We further modified the simulator to explore Configanator performance when different NC techniques are used. We observe that Configanator out-performs the rest for the majority of the pageloads. The prefix and CDN based approaches either do not account for network dynamics or overfit to specific regions respectively. Latency driven performs slightly better but ignoring the important metrics, like packet loss and bandwidth, degrades its effectiveness.

F.7 PLT Variability

PLT measurements are inherently noisy [96] and the variability in PLT can disrupt the learning algorithm's model, e.g., GP is sensitive to noise [78]. Using data from a web performance observability company (NewRelic [71]), we modeled PLT variability distribution and used it to introduce variability in testbed-generated PLT-Tensor. Figure 27a plots a PDF of the variations with x-axis as the mode PLT multiple (x-axis is PLT normalized by mode PLT). We fit an Erlang curve to the observed PDF, owing to its right-skewed, long-tail nature. Using PLT from PLT-Tensor as the mode PLT (since mode PLT is the most stable PLT measurement), the PDF is used to calculate the noisy PLT observed by a real-world user.

Figure 27b plots the extent to which Configanator decisions (in face of PLT noise) are optimal, compared to the case when there is no noise (Configanator-NoVari). A proximity score of 1 indicates that Configanator decisions stay exactly the same for both (noise, no-noise) cases. Leveraging the *PerfMemory* knob, we test 2 scenarios with different length of historical memory. Configanator uses this historical memory to amortize the impact of any sudden change in performance metrics. We observe Configanator's decisions to slightly deteriorate in face of noise. However the extent is mild at worst — with the system still assigning the optimal decisions more than 95% at median.

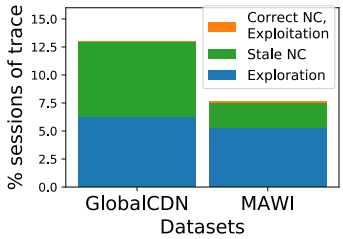


Figure 28: % sessions with PLT degradation.

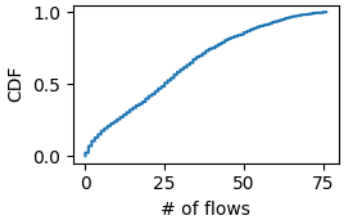


Figure 29: Number of flows through access link

F.8 Dissecting PLT Degradation

As shown in Figure 5, all algorithms result in some PLT degradation. Figure 28 plots the percentage of sessions that faced PLT degradation and further divide them into the root-causes. Our observations are as follows: (i) During exploration, multiple configurations are tested and may result in degradation. Around 5-6.2% of the sessions in two of the datasets are such exploration steps. (ii) As network conditions change over time, Configanator's estimate of historical network characteristics for an IP-prefix may diverge from the actual network. The stale information is used for classifying the connection into an NC and predicting the optimal configuration. Due to the global nature of GlobalCDN dataset, we observe a higher network churn, with 6.6% of total sessions resulting in PLT degradation due to stale NC. Only a small proportion of sessions in MAWI dataset ($\sim$0.1%) resulted in PLT degradation with correct NC view, indicating that the exploitation arm momentary got stuck at a sub-optimal configuration.

F.9 CM Design Choices

Configuration Manager Design: CM can run locally in a PoP or centrally within a data-center [46], trading-off between data-size to learn and the speed to react to changes. We evaluate both scenarios in our simulator: In the local design, there's a separate CM for each trace, while for the global case there's a single CM for all traces. To simulate each scenario, we vary the latencies between CM and the web servers. We observe that while the global CM is able to make slightly better predictions at the tail (2% better than local), the difference at median is negligible. Despite the larger data set, global CM is not significantly better due to distinctly diverse network conditions across regions (only 17% NCs are common in U.S and Japan traces).

Frequency of model updates: Next, we analyze the impact of updating our performance model less frequently: we explore a range of values from every 2 minutes [12] up to every day. We observed performance to stay relatively stable at the median, whereas hourly or lower update intervals result in $\sim$8% better improvement at tail, than a per-day granularity.

[12] It takes $\sim$2 minutes to update the models for 10K sessions.

F.10 Flows Through Access Link

We use packet trace from [115] to measure the typical number of TCP flows through an access link. Figure 29 presents the number of TCP flows with at least 10Kb data transferred, in a 60s time interval. On the median 60s time interval, we observe around 25-30 flows competing through the access link.

F.11 Additional Micro-benchmarks

In addition to the system benchmarks in § F.9, we also evaluated two alternate design choices: VMs and LD_Preload. For VMs, we used one VM for each configuration and used Open vSwitch (OVS) [43] for routing flows to the appropriately configured VM. We explored the use of *LD_Preload* to intercept system call and tuned socket using *setsockopt()*. In comparing both choices with Configanator, we observed that the VM-based approach introduced a 20% increase in latency where as the *LD_Preload* introduced a much smaller latency of 2.2%. We also observed overheads for CPU and Memory utilization: the VM-based approach introduced 30% (memory taken by the guest OS) while *LD_Preload* introduced a 5% increase.