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

Usama Naseer, Theophilus A. Benson

BROWN

Web Performance Matters!

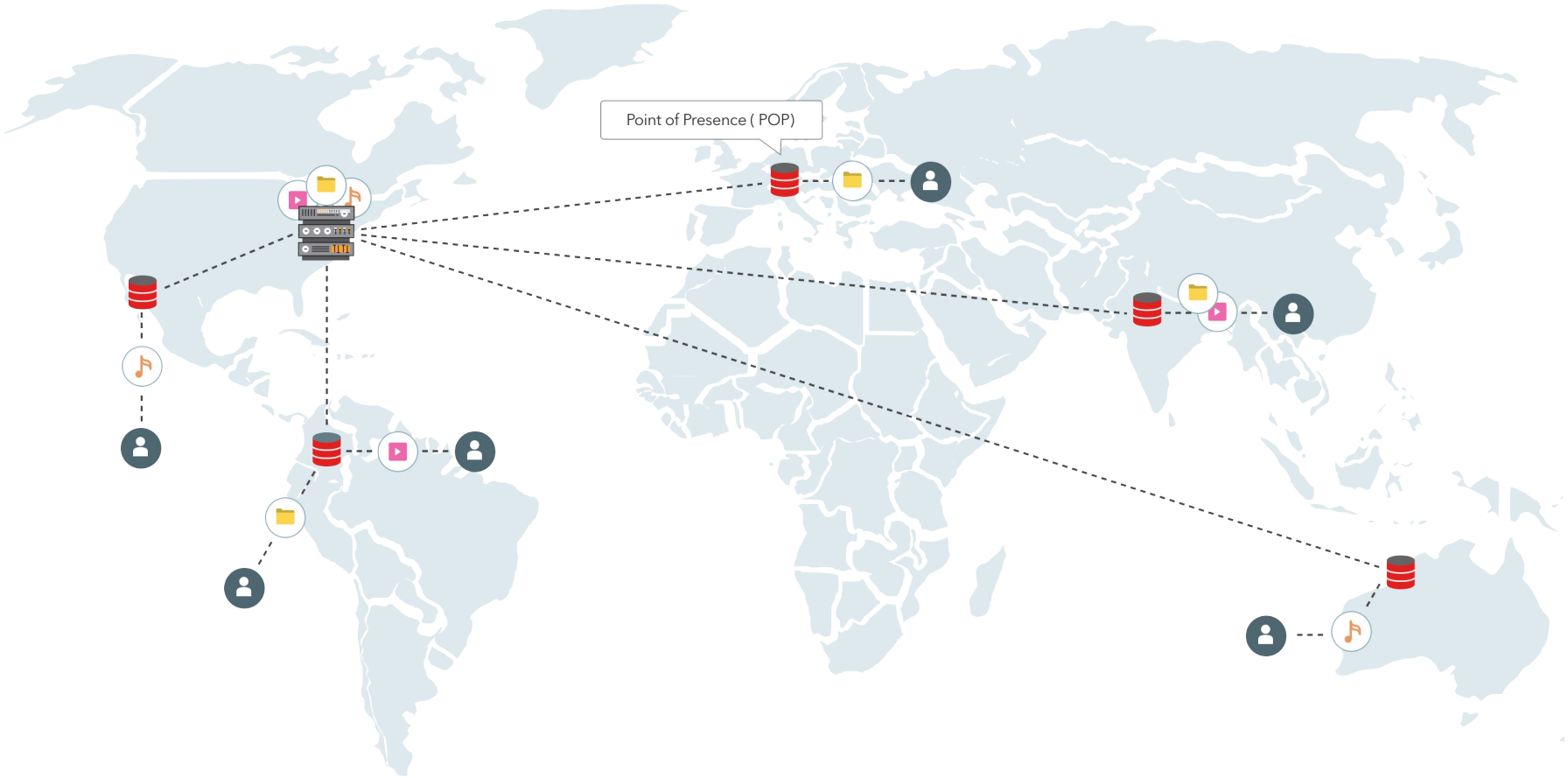
Performance of digital services have a direct impact on businesses & society.

Increases user engagement

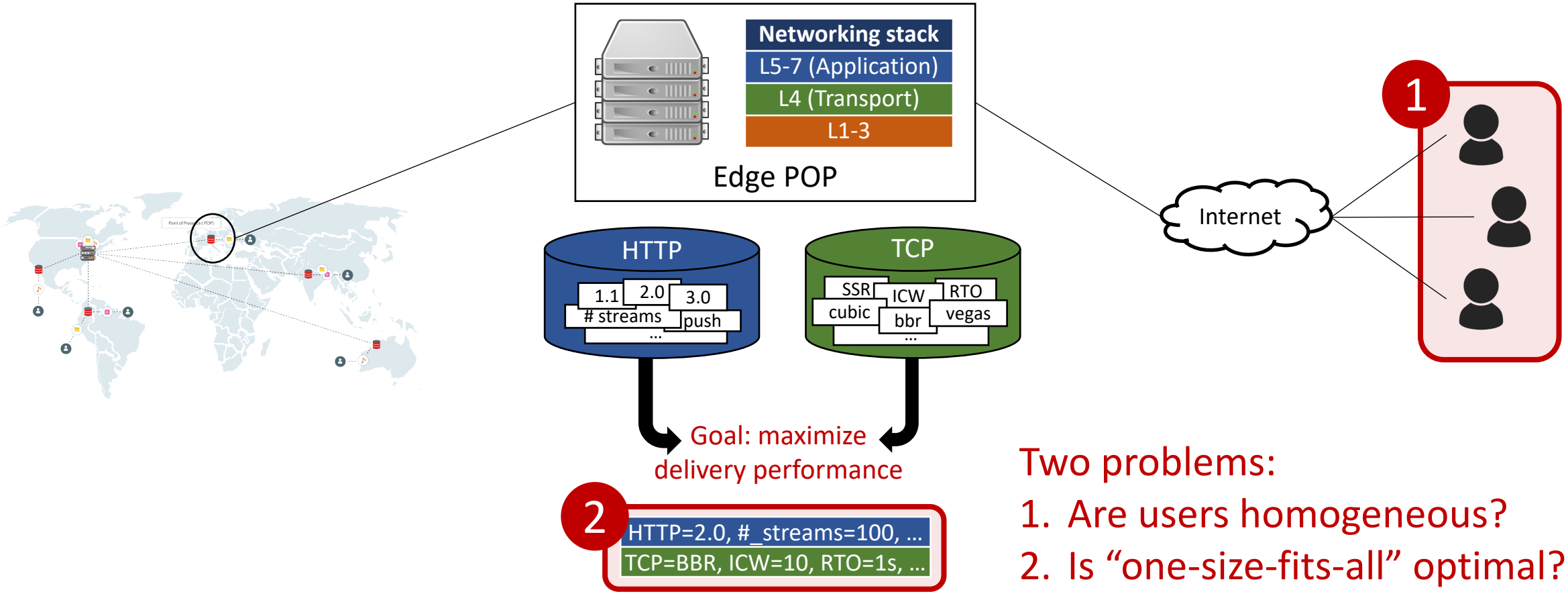
Growth in revenue

Improves productivity

Background



Background

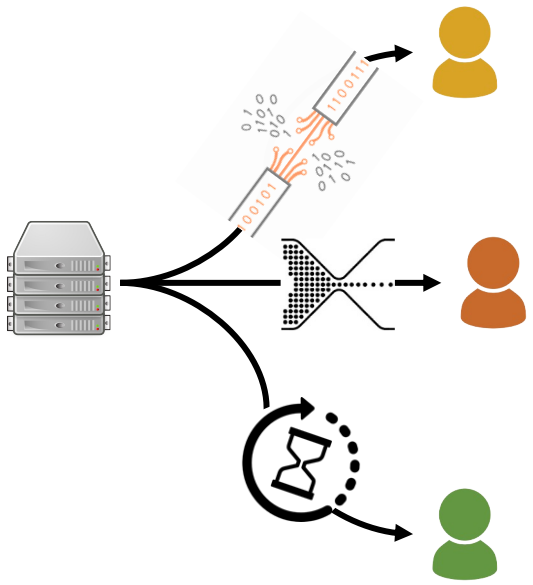


Two problems:

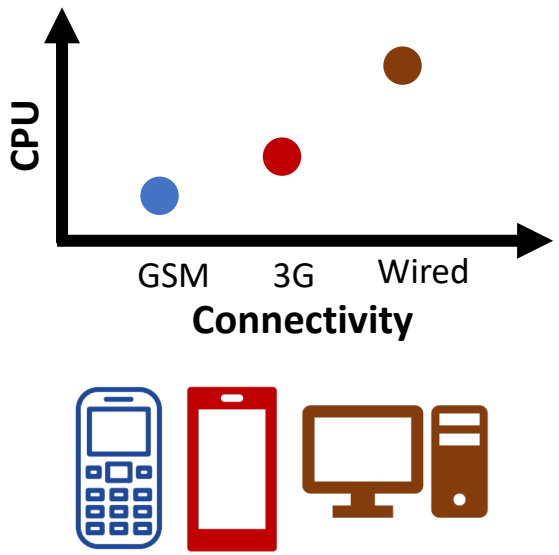
1. Are users homogeneous?
2. Is “one-size-fits-all” optimal?

User Heterogeneity & Performance Sensitivity.

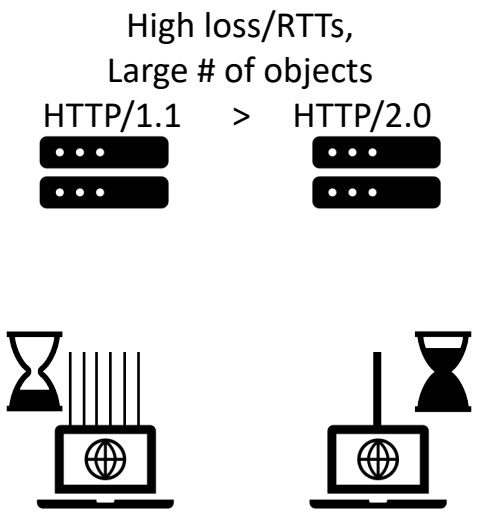
- Heterogeneity due to network and device diversity.
- Performance contingent on networks, device dynamics, and website complexity.



Different congestion control models (delay/loss/bottleneck-bw) [Yan et al. ATC'18]



Impact of device capabilities [Ahmad et al. IMC'16]

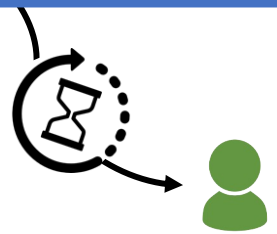


Impact of network & website complexity [Wang et al., NSDI'14]

User Heterogeneity & Performance Sensitivity.

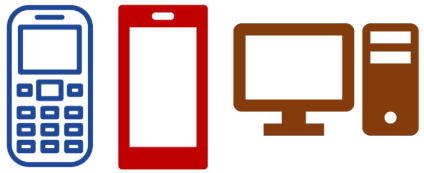
- Heterogeneity due to network and device diversity.
- Performance contingent on networks, device dynamics, and website complexity.

One set of protocol configurations is sub-optimal for diverse connections.

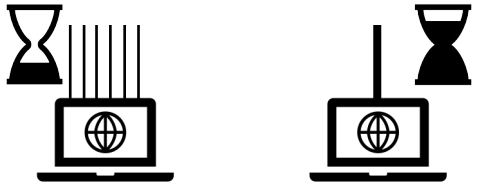


Different congestion control models (delay/loss/bottleneck-bw) [Yan et al. ATC'18]

Connectivity



Impact of device capabilities [Ahmad et al. IMC'16]



Impact of network & website complexity [Wang et al., NSDI'14]

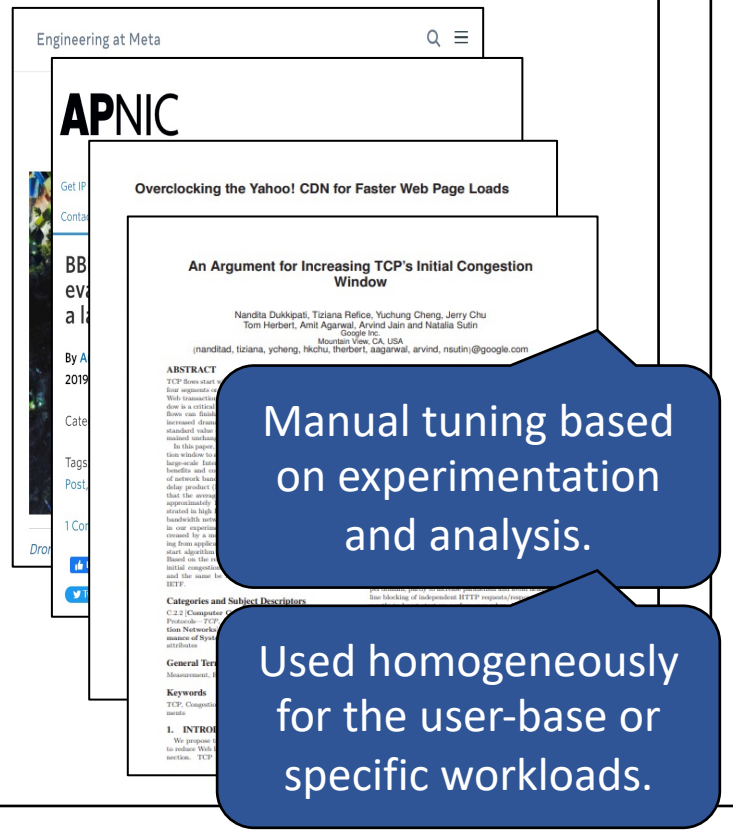
How to dynamically tune the networking configurations to maximize performance for the diverse connections?

Traditional Approaches for Selecting Configurations

Default

```
net.ipv4.tcp_congestion_control = cubic
net.ipv4.tcp_low_latency = 0
net.ipv4.tcp_autocorking = 1
default via IP dev eth0 initcwnd 10
Protocols http/1.1
```

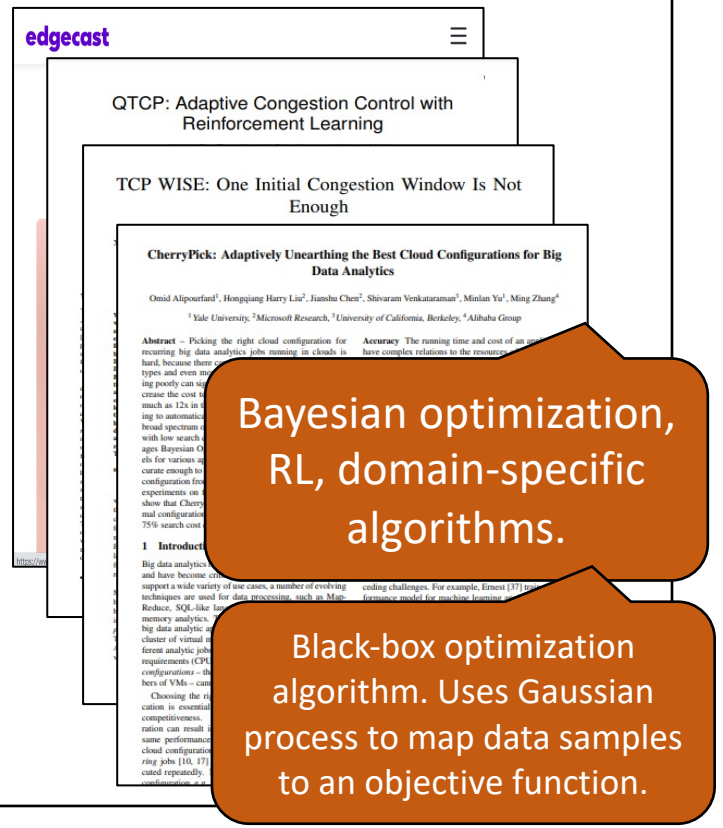
HandPicked



Manual tuning based on experimentation and analysis.

Used homogeneously for the user-base or specific workloads.

Dynamic



Bayesian optimization, RL, domain-specific algorithms.

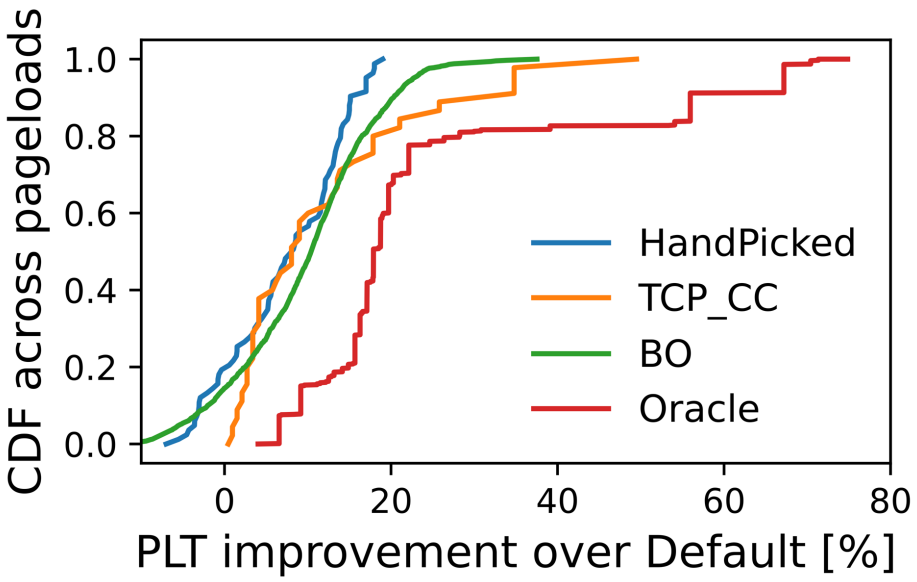
Black-box optimization algorithm. Uses Gaussian process to map data samples to an objective function.

Are Traditional Approaches Optimal?

- Emulated diverse traces from a CDN in local testbed.
- Brute-force exploration of TCP and HTTP configuration space.
- *Oracle:* Selects optimal configurations that minimizes page load time (PLT).

Single configuration dynamically tuned for the connections.

Pre-computed configs. statically applied to diverse connections.



BO -> sub-optimal convergence, no cross-connection learning.

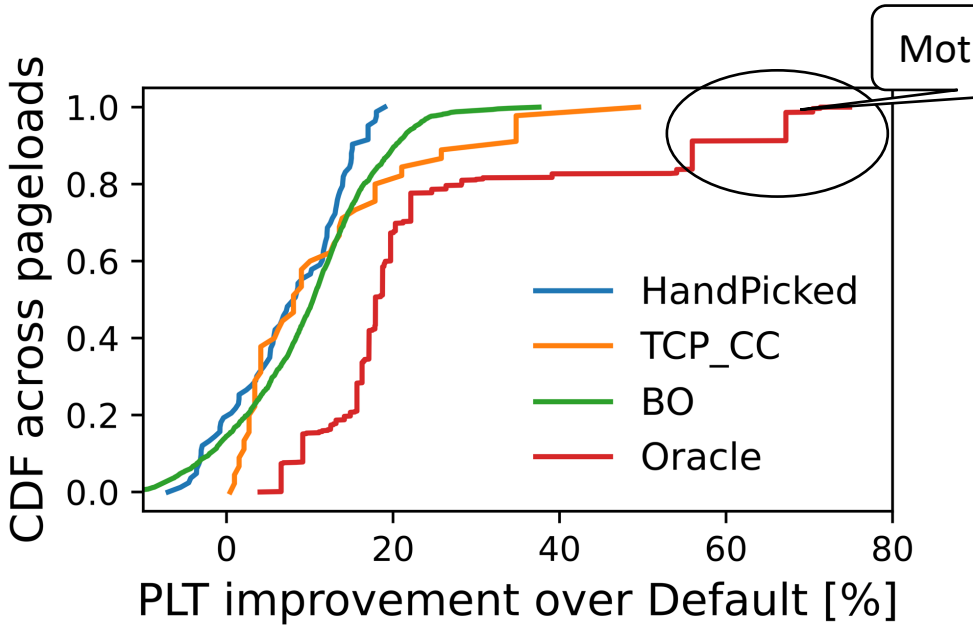
Per-connection optimal configs. -> significant performance gains.

Are Traditional Approaches Optimal?

- Emulated diverse traces from a CDN in local testbed.
- Brute-force exploration of TCP and HTTP configuration space.
- *Oracle:* Selects optimal configurations that minimizes page load time (PLT).

Single configuration dynamically tuned for the connections.

Pre-computed configs. statically applied to diverse connections.



Motivates dynamic tuning.

BO -> sub-optimal convergence, no cross-connection learning.

Per-connection optimal configs. -> significant performance gains.

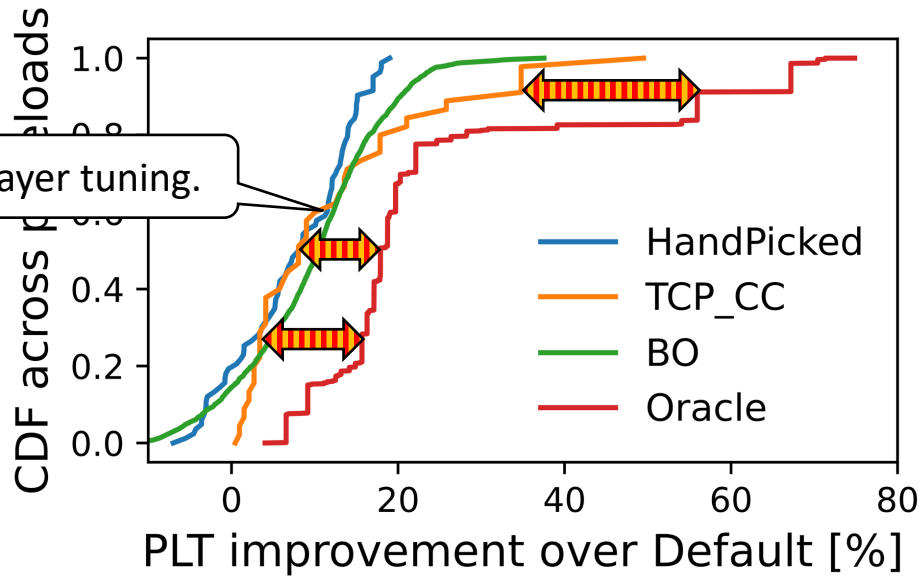
Are Traditional Approaches Optimal?

- Emulated diverse traces from a CDN in local testbed.
- Brute-force exploration of TCP and HTTP configuration space.
- *Oracle:* Selects optimal configurations that minimizes page load time (PLT).

Single configuration dynamically tuned for the conne

Motivates cross-layer tuning.

Pre-computed configs. statically applied to diverse connections.



BO -> sub-optimal convergence, no cross-connection learning.

Per-connection optimal configs. -> significant performance gains.

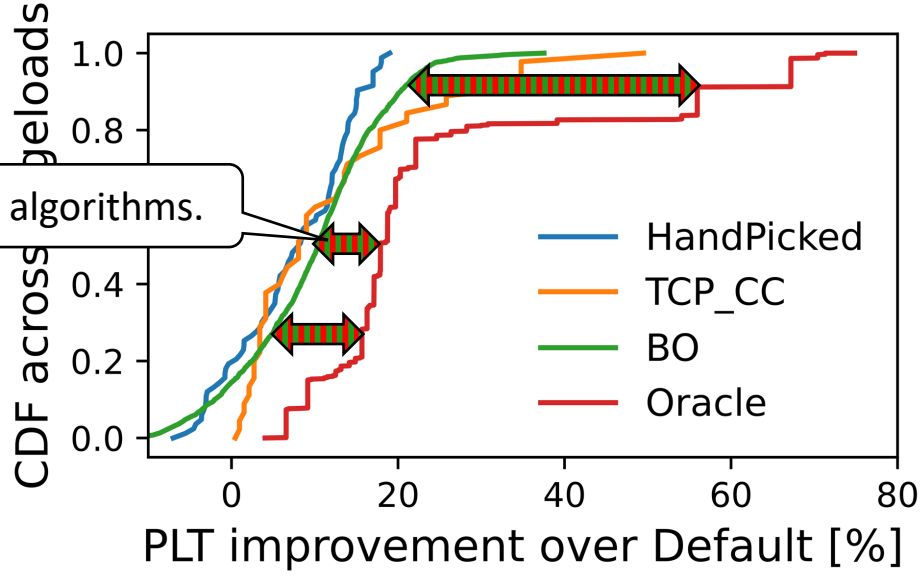
Are Traditional Approaches Optimal?

- Emulated diverse traces from a CDN in local testbed.
- Brute-force exploration of TCP and HTTP configuration space.
- *Oracle:* Selects optimal configurations that minimizes page load time (PLT).

Single configuration dynamically tuned for the conne

Pre-computed configs. statically applied to diverse connections.

Motivates better algorithms.

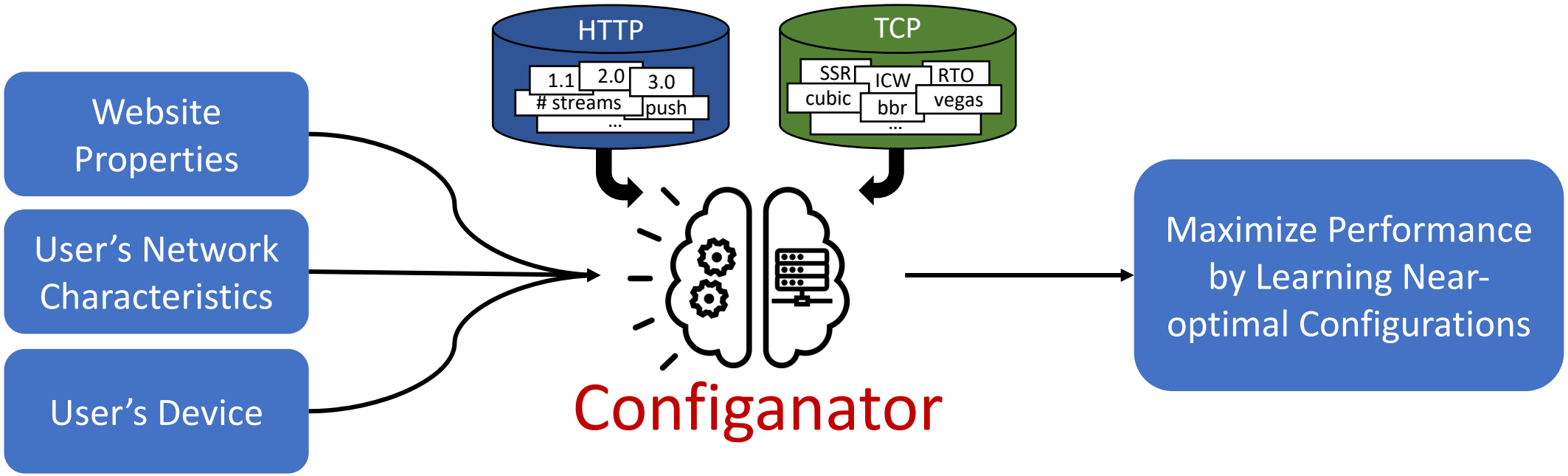


BO -> sub-optimal convergence, no cross-connection learning.

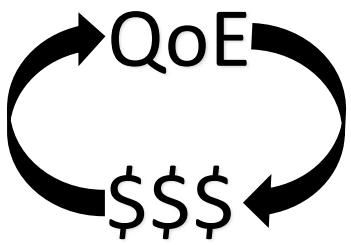
Per-connection optimal configs. -> significant performance gains.

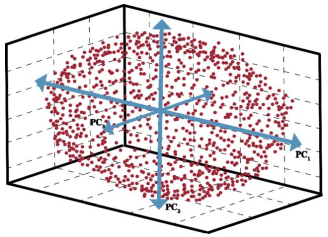
Configanator

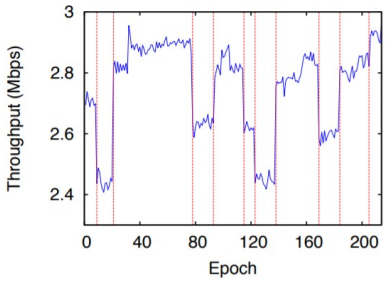
Optimizes web performance by systematically reconfiguring network stack in a principled manner.

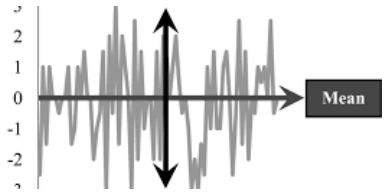


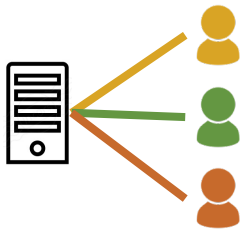
Challenges with Configuration Tuning











<u>QoE Cost</u>
Bad configurations hurts revenue.

<u>High dimensionality</u>
Connection types, devices & websites.

<u>Network dynamics</u>
Network & performance changes over time.

<u>Noise & variability</u>
Inherent noise in performance signals.

<u>System limitations</u>
Low-overhead, fine- grained tuning.

Algorithm design

System design

Network dynamics figure credits: Zahaib Akhtar

Algorithmic Design

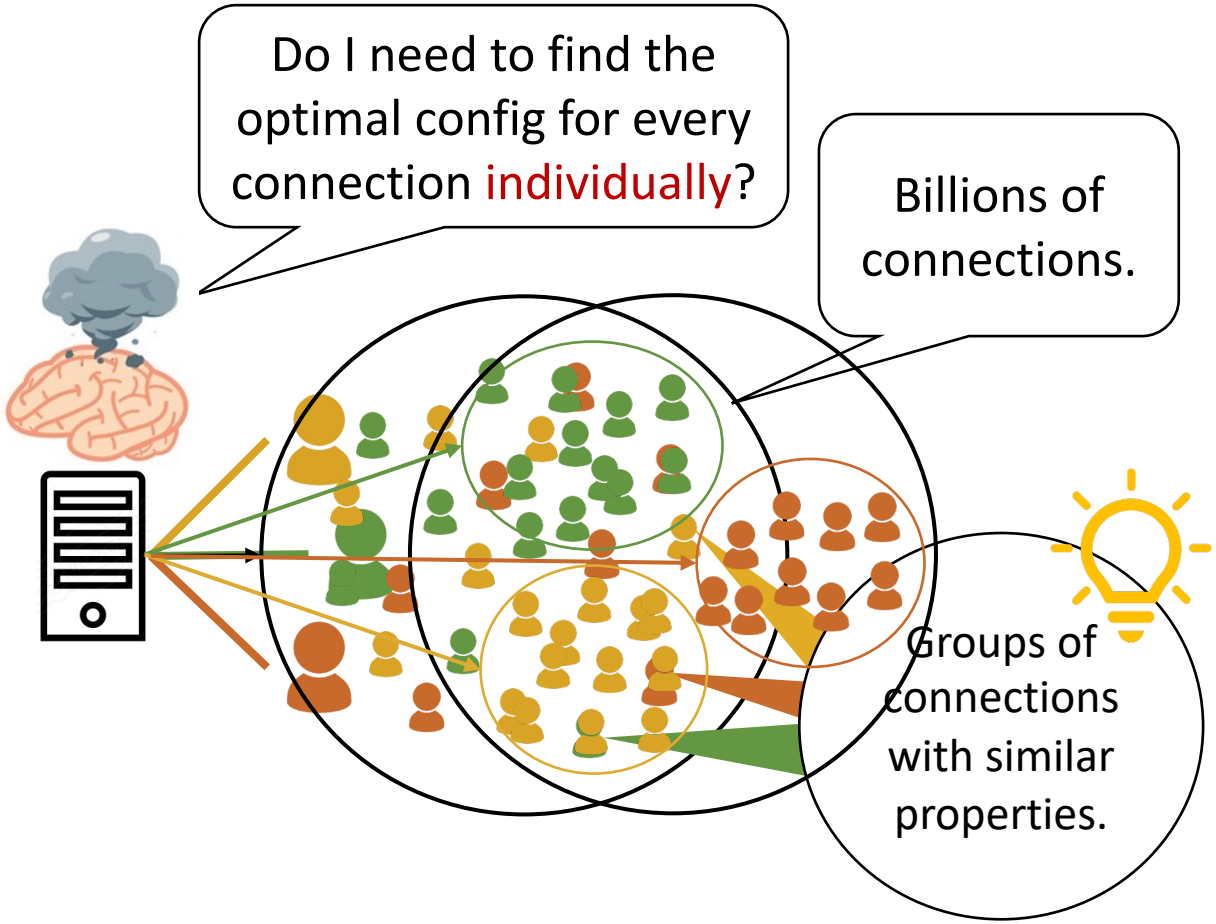
Tuning and exploration granularity?

Coarse-granularity -> ASN, POP, prefixes?

- ✓ Amortizes QoE cost.
- ✗ Fine-grained diversity.
- ✗ High dimensionality.

Network Classes (NC) -> connections with similar properties.

- ✓ Amortizes QoE cost.
- ✓ Fine-grained diversity.
- ✓ High dimensionality.



Algorithmic Design

Offline approach not representative of high dimensional Internet.

Build performance models for the high-dimensional web in a low-cost, online manner.

Exploration arm
Bayesian Optimization (Gaussian process)

- ✓ Efficient exploration.
- ⊘ Sub-optimal convergence (measurement noise).
- ⊘ Sub-optimal exploitation (model isolation).
- ↓ Up to 2X lower median improvement.

Exploitation arm
ML techniques (decision trees)

- ✓ Efficient exploitation.
- ⊘ No guided exploration (high QoE-cost).
- ↓ Up to 40% lower tail improvement.

C1 → Contextual Multi-armed Bandit ← C2

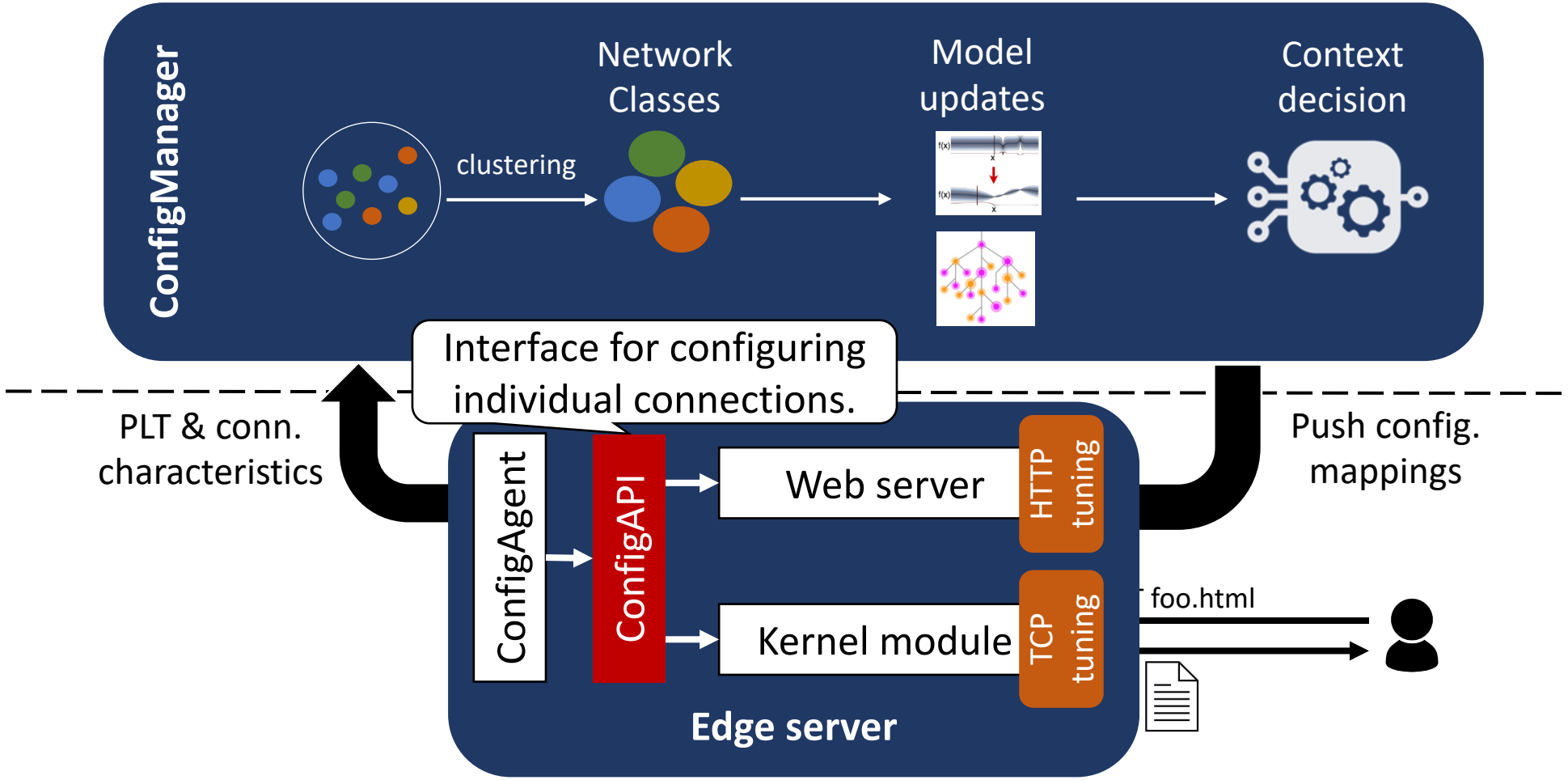
C3 → Contextual Multi-armed Bandit

Resampling

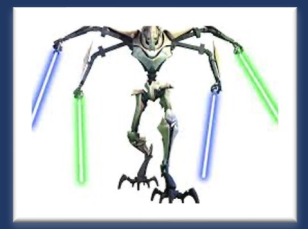
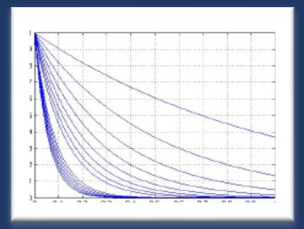
Context decision
Probabilistic gain of exploring a new configuration?

expected_improvement from *GP*

Configanator Workflow



Evaluation

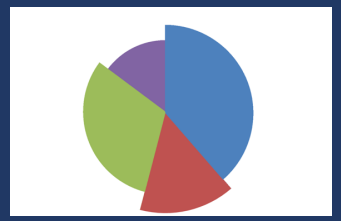



Algorithm design (benefits of arms, convergence, NCs)

Performance improvements.

Fairness implications.

System overheads.



Dissection of improvements, deployment considerations.

Evaluation

Algorithm
arms, co

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, 29]. Yet, uniformly improving web performance is becoming increasingly challenging due to the growing disparity in the network conditions (e.g. bandwidth, RTT) [3, 35, 118, 132] and end-user devices [91, 92, 106, 131]. 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 [30], BBR [23], QUIC [50], etc.

The optimal choice of configurations is contingent on the network infrastructure [3, 35, 75, 96, 118, 132, 139], website complexity [20, 21, 93, 131, 133], and end-user devices [1, 92, 106]. 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, 91], a majority of CDNs continue to employ a "one-size-fits-all" [48] approach to configuring their edge servers, which results in sub-optimal performance [3, 35, 132] and high tail-latency in certain regions [139].

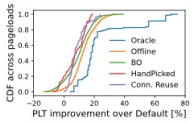


Figure 1: Comparison of various tuning techniques.

1.1 Configuration Tuning Status-Quo

Most attempts to tackle this growing diversity involve manually analyzing the performance of configuration options across different regions [48], devices [1], or websites [108, 132]. While several CDNs expose configuration knobs to their customers [39, 44], 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* [102] (e.g., CherryPick [4]) a statistical

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 [48, 90, 108, 140]. 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.

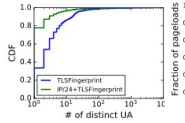


Figure 15: Relationship between TLS fingerprint and User-Agent.

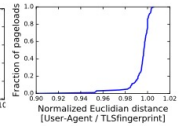


Figure 16: Comparison of device identifier's impact on NCs.

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).

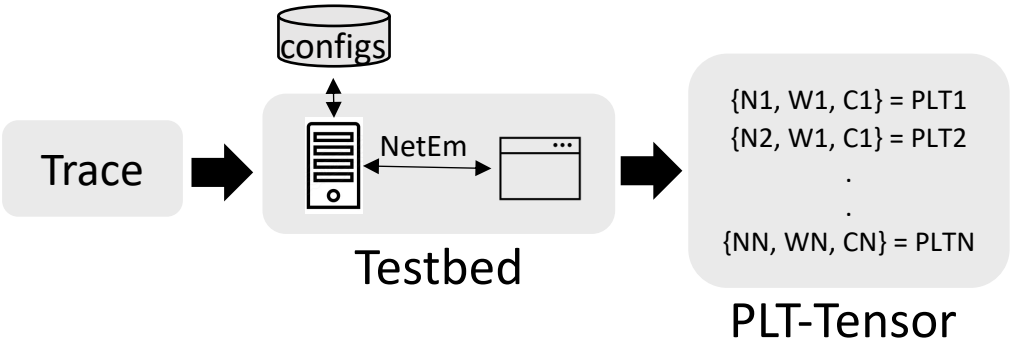
C Passively Recording Network Conditions

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 [115]. 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.

Evaluation Setup

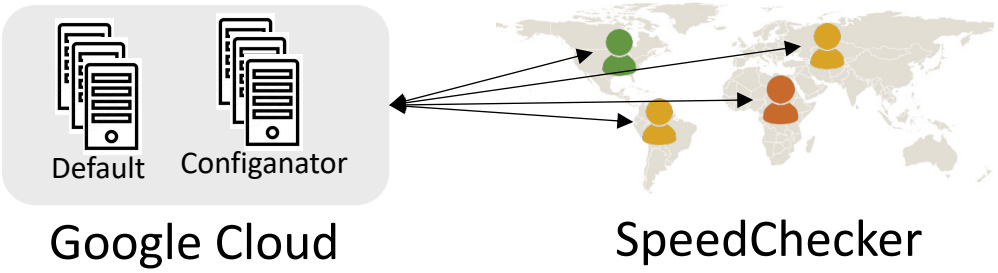
Trace-driven simulation

Traces from multiple regions.
- US (CAIDA, FCC)
- Japan (MAWI)
- Global (CDN trace, Pantheon)

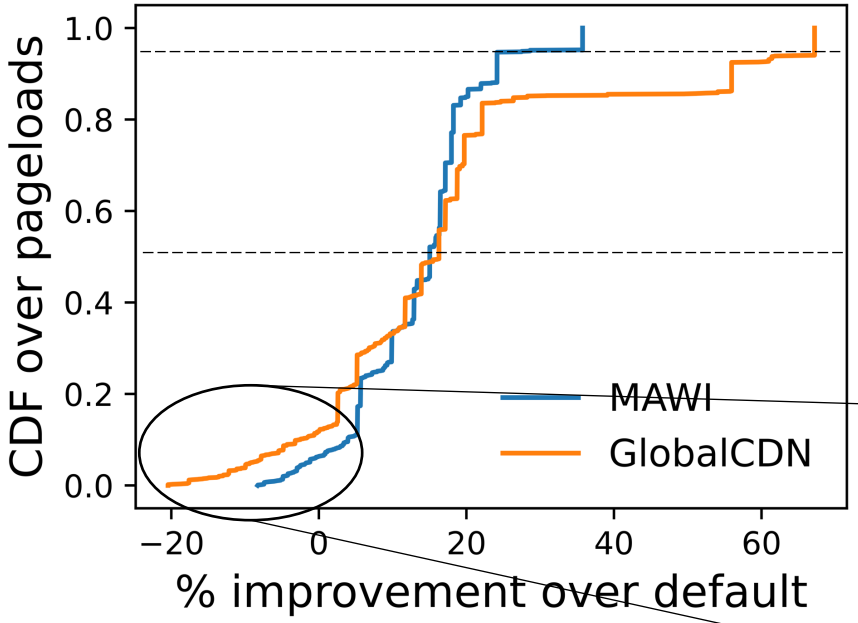


Live deployment

Experiments in-the-wild.
- Servers (Google cloud in US)
- Users (spread across globe)

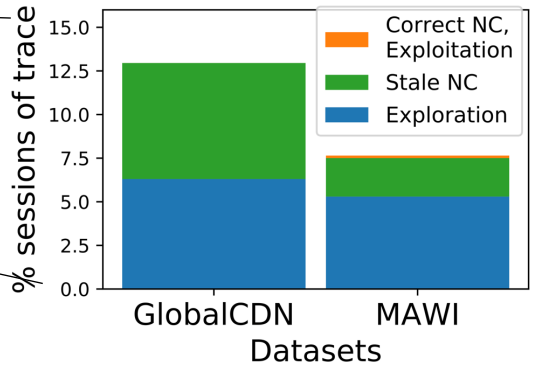


Performance Improvements



36-67% improvement.
Low BW, low-high RTT/loss networks, content-rich sites.

15-17% improvement



Live deployment

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

Related Works

- Web optimizations.
  - Protocol sensitivity (e.g., ICW, HTTP), device bottlenecks (e.g., mobile CPU), optimized delivery (e.g., object scheduling, prioritization).
  - Exploit heterogeneous protocol configuration to improve delivery.
- Self-tuning systems.
  - Video (e.g., ABR tuning), analytics (e.g., data-bases, Hadoop), cloud computing (e.g., cluster management).
  - Online, domain-specific algorithm to tackle dynamicity & multi-dimensionality of the Internet.
- Transport optimizations.
  - Novel congestion controls (e.g., PCC, TCP Ex Machina), transport tuning (e.g., CC, ICW), congestion management, new transports (e.g., QUIC).
  - Auto-tuning system for cross-layer protocol configurations.

Conclusion + Future Work

Customizable protocols

Algorithm to tackle diversity.

Contextual MAB

Cross-layer protocols

Broader configurations and flexibility.

QUIC

Transport loss

Web priorities

Infrastructure support