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

Usama Naseer, Theophilus A. Benson
Web Performance Matters!

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

- Increases user engagement
- Growth in revenue
- Improves productivity
Background
Background

Goal: maximize delivery performance

HTTP=2.0, #_streams=100, ...
TCP=BBR, ICW=10, RTO=1s, ...

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]

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`

- **HandPicked**
  - Protocols `http/1.1`
  - 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.**

**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).

- Pre-computed configs. statically applied to diverse connections.
- Single configuration dynamically tuned for the connections.
- BO -> sub-optimal convergence, no cross-connection learning.
- Per-connection optimal configs. -> significant performance gains.

Motivates cross-layer tuning.
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).

Motivates better algorithms.

Pre-computed configs. statically applied to diverse connections.

Single configuration dynamically tuned for the connection.

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.

Website Properties
User’s Network Characteristics
User’s Device

Maximize Performance by Learning Near-optimal Configurations
Challenges with Configuration Tuning

QoE Cost
Bad configurations hurts revenue.

High dimensionality
Connection types, devices & websites.

Network dynamics
Network & performance changes over time.

Noise & variability
Inherent noise in performance signals.

System limitations
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.

Billions of connections.

Do I need to find the optimal config for every connection individually?

Groups of connections with similar properties.

15
Algorithmic Design

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.

Contextual Multi-armed Bandit

Context decision
Probabilistic gain of exploring a new configuration?

Resampling

Offline approach not representative of high dimensional Internet.

expected_improvement from GP
Configanator Workflow

1. **ConfigManager**

2. **Network Classes**
   - Clustering

3. **Model updates**

4. **Context decision**

5. **Interface for configuring individual connections.**

6. **PLT & conn. characteristics**

7. **Push config. mappings**

8. **ConfigAgent**

9. **ConfigAPI**
   - Web server
   - HTTP tuning
   - TCP tuning
   - Kernel module

10. **Edge server**

11. **foo.html**
Evaluation

- Algorithm design (benefits of arms, convergence, NCs)
- Performance improvements.
- Fairness implications.
- System overheads.
- Dissection of improvements, deployment considerations.
Evaluation

Figure 1: Comparison of various tuning techniques.

Algorithm, arms, co...

Table 4: Similar leads

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

1 Introduction

With page performance significantly impacts the revenue of content distribution networks (CDNs) (e.g., Facebook, Amazon, or Google), studies showing that a 100ms decrease in page load time (PLT) can lead to a 1.8% decrease in revenue for the retail sales (14, 12). Yet, uniformly improving web performance is becoming increasingly challenging due to the growing disparity in the network conditions (e.g., bandwidth, RTTs (13, 15, 16, 13) and network overheads (11, 11)). To address this disparity and improve the quality of experience (QoE), the networking community is continuously developing new protocols and configuration parameters for web servers (AES, edge servers, etc.) (e.g., PCC/HTTP/2, QUIC) (10).

The optimal choice of configuration is contingent on the network infrastructure (1, 15, 17, 16, 16, 16, 16) and user devices (13, 13, 13). Furthermore, innovations along any one of these dimensions will have changes to initial parameters and new protocols. Although different regions and ISPs have radically different networking infrastructure and mobile devices (11, 15), a majority of CDNs continue to employ a “one-size-fits-all” (19) approach to configuring their edge servers, which results in sub-optimal performance (15, 15, 15) and high tail latency in certain regions (15).

1.1 Configuration Tuning Status-Quo

Most strategies to tackle this growing disparity involve manually assessing the performance of configuration options across different regions (16), devices (11), or websites (15, 15, 15, 15). While several CDNs expose configuration limits to their customers (17, 15), it is challenging to take full advantage of these APIs due to the lack of automated learning tools for server tuning.

This paper focuses on using a broad set of configuration limits across the network (e.g., congestion control algorithms and application layers, e.g., HTTP versions as highlighted in [17, 15]). Next, we illustrate the challenges and benefits of dynamic tuning network configurations.

Challenges in tuning: 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 (12, 12). Specifically, Response Time (RTT) (12, 12), e.g., (CherryPick) is a minimal...

Usama Nasser
Brown University
Thothelia. A. Benson
Brown University

Table 4: Similar leads

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

A Fingerprinting Configurations

Our fingerprinting techniques are inspired from recent works (20, 10, 10, 10). Our test inter-protocol with TLS and different implementations are the following ways: (1) HTTP configurations are visible to clients during the connection setup and are fingerprinted from the server response, (2) TCP configurations like DNS are stripped from the packet headers, (3) TCP/HTTPD is not implemented a load balancing tool, (4) by not acknowledging RST packet back to the server, and measuring the time it takes the server to terminate the TCP/HTTPD, and (5) timing a request to the server and the number of packets sent by the server after RST is measured. Further, we also measured a higher number of packets from the server. We used AWS in a regional AB (n) as the source for fingerprinting the configuration.

B TLS Fingerprinting for Device Identification

Recall that instead of the traditional User-Agent string, Configurator uses TLS fingerprinting for device identification as a dynamic system identifier in early stage of the connection (prior to the HTTPS version negotiation through ALPN). To evaluate its efficacy, we leverage a dataset from GlobalDNS, containing 14M requests. The dataset consists of server logs and captures User-Agent strings from HTTPS GET requests and the TLS fingerprint of the respective connections. The dataset includes 14.3K unique User-Agent strings and 13.2K unique TLS fingerprints.

Figure 15 shows the number of unique User-Agents (UA) that maps 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 4% of the fingerprints, with the rest mapping to about 70% UAs. We observe that complementing the TLE fingerprints with one-to-one IP maps helps in improving the accuracy, while the number of fingerprint mapping to a single (UA, and 90% mapping to multiple devices) remains high. Further, we observe that the case where a single fingerprint maps to multiple UA strings, there are only minor differences, e.g., different browser versions, different in OS’s source version (Ubuntu 6.0, 6.0, 6.0).

In Figure 16, we further compare the two device identification techniques for choosing similar connections together. Using a dataset of 18M PLT measurements from GlobalDNS, we use our Network Classification using either User-Agent or TLE fingerprints as the basic device for device identification. We compare the machine learning of each connection PLT from the client’s context and the figure gives the ratio of 0.2. We observe that the ratio is between 0.05 and 0.01 for the overwhelming majority of the tuples, indicating that the two techniques 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 HTTPS version, whereas User-Agent can only be inferred through HTTPS requests headers (received after HTTP version negotiation).

C Passively Recording Network Conditions

Configurator passively collects go-purpose and packet loss rates for the IP prefix (24) and builds a historical archive (3, 3, 3). When an SSL context, Configurator uses the handshake RRT and looks up the go-purpose and packet loss rates from the recent session for the IP prefix (24) and classifying the class into Network Class. Configurator parses the User-Agent string, Apache logs, and Firewall logs to classify the User-Agent string. However, we observe that the user-agent string can contain the device model and the operating system name. Furthermore, the feature size is likely to be small due to having less than 30 features.

Figure 17 shows the number of unique User-Agent strings that maps 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 4% of the fingerprints, with the rest mapping to about 70% UAs. We observe that complementing the TLE fingerprints with one-to-one IP maps helps in improving the accuracy, while the number of fingerprint mapping to a single (UA, and 90% mapping to multiple devices) remains high. Further, we observe that the case where a single fingerprint maps to multiple UA strings, there are only minor differences, e.g., different browser versions, different in OS’s source version (Ubuntu 6.0, 6.0, 6.0).

In Figure 16, we further compare the two device identification techniques for choosing similar connections together. Using a dataset of 18M PLT measurements from GlobalDNS, we use our Network Classification using either User-Agent or TLE fingerprints as the basic device for device identification. We compare the machine learning of each connection PLT from the client’s context and the figure gives the ratio of 0.2. We observe that the ratio is between 0.05 and 0.01 for the overwhelming majority of the tuples, indicating that the two techniques 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 HTTPS version, whereas User-Agent can only be inferred through HTTPS requests headers (received after HTTP version negotiation).
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)

Trace

\[
\begin{align*}
\{N1, W1, C1\} &= PLT1 \\
\{N2, W1, C1\} &= PLT2 \\
\{NN, WN, CN\} &= PLTN
\end{align*}
\]

NetEm

PLT-Tensor

Google Cloud

SpeedChecker
Performance Improvements

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

15-17% improvement

Live deployment

<table>
<thead>
<tr>
<th>Pageload</th>
<th>PLT diff. (ms)</th>
<th>% imp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>p25</td>
<td>671</td>
<td>13.7</td>
</tr>
<tr>
<td>p50</td>
<td>767</td>
<td>14.6</td>
</tr>
<tr>
<td>p75</td>
<td>1219</td>
<td>21.5</td>
</tr>
<tr>
<td>p95</td>
<td>3797</td>
<td>26.3</td>
</tr>
</tbody>
</table>
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.
- Cross-layer protocols
- Infrastructure support

- Contextual MAB
- Broader configurations and flexibility.
- Transport loss
- Web priorities
- P4