Configtron: Tackling network diversity with heterogeneous configurations.

Usama Naseer *and* Theophilus Benson
Duke University

Abstract

The web serving protocol stack is constantly changing and evolving to tackle technological shifts in networking infrastructure and website complexity. For example, Cubic to tackle high throughput, SPDY to tackle loss and QUIC to tackle security issues and lower connection setup time. Accordingly, there are a plethora of protocols and configuration parameters that enable the web serving protocol stack to address a variety of realistic conditions. Yet, despite the diversity in end-user networks and devices, today, most content providers have adopted a "one-size-fits-all" approach to configuring user facing web stacks (CDN servers).

In this paper, we illustrate the drawbacks through empirical evidence that this "one-size-fits-all" approach results in sub-optimal performance and argue for a novel framework that extends existing CDN architectures to provide programmatic control over the configuration options of the CDN serving stack.

1 Introduction

With the internet entering the zettabyte era and connecting billions of users, there is a huge disparity in the network conditions (bandwidth and loss rates) faced by end-users [5]. To address this disparity and improve quality of experience (QoE), online service providers (OSP) are constantly developing novel protocols and configuration standards for their content distribution network or point of presence servers (CDN). Over the last several decades, the networking community has developed a broad range of protocols across various layers of the stack from New Reno, Vegas, Compound TCP to Cubic, from HTTP1.1, and SPDY to HTTP2.

While the community continues to develop new protocols and establish new configuration standards, there is an erroneous assumption that the newer protocols and configuration standards are strictly better than the older ones. For example, Cubic and HTTP2 are wildly deployed despite proven [31, 8, 25] evidence that these protocols are sub-optimal under certain conditions (§ 2.1).

The optimal choice of protocol and parameters is contingent on the network infrastructure [31, 8, 20], website complexity [4, 22], and end-user device. Furthermore, the constant evolution of networking infrastructure, end-user devices, and web complexity results in the invention of new protocols and the re-evaluation of old configuration parameters. Although different regions and ISPs leverage radically different infrastructures and host radically different devices, CDNs continue to employ a "one size fits all" which results in sub-optimal performance in some regions [2]. Motivated by this sub-optimal performance, online service providers are taking drastic steps to educate and motivate their developers to explicitly tackle underlying network and end-user heterogeneity. For example, Facebook instituted "2G Thursday" — on Thursday all traffic is throttled to 2G speeds forcing developers and network operators to tackle issues faced by users on 2G networks [9].

In this paper, we eschew the notion of a "one size fits all" approach to protocol and configuration selection for CDN servers and instead argue for a "curated" approach to protocol selection and configuration. In particular, we argue that CDN servers should be configured and setup with the optimal protocol and configuration parameters for serving each connection. For example, a CDN server serving high loss, low bandwidth connections may employ a lower initial window size than a server serving low loss, high bandwidth connections.

To this end, we propose a framework for practically improving end-user performance by introducing heterogeneity into the CDN server protocol configuration in a principled manner. We argue for introducing a simple but standard interface to the CDN server's network stack that exposes existing heterogeneity (e.g. TCP versions, HTTP protocol, or Peering link) and enables remote configuration. Additionally, our framework includes a learn-

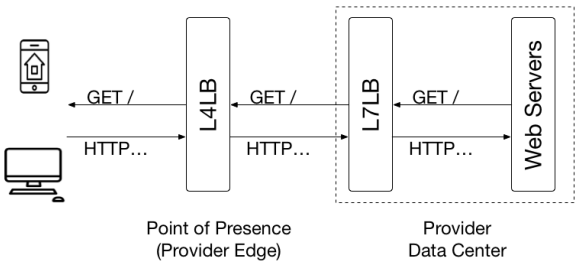

Figure 1: Infrastructure for OSPs (e.g. Facebook [30]).

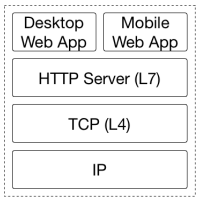

Figure 2: CDN server's networking stack.

ing entity that determines the optimal configuration for each end-user based on a combination of emulations and passive measurements. Together our interface and the learning entity enables a CDN to systematically introduce heterogeneity into the server's network stack in a manner that systematically improves end-users performance.

This paper takes the first step towards realizing our system, Configtron, for reconfiguring CDN server networking stacks. Specifically, we make the following contributions:

- First, we present a clear demonstration that one size configuration, does not fit all network conditions and web site complexities and moreover no one configuration is strictly better than another, thus motivating the case for heterogeneity within the CDN infrastructure.
- Second, the design of a framework for systematically learning optimal configurations and practically reconfiguring CDN web servers without loss of functionality.
- And lastly, a strawman implementation of our Configtron, demonstrating the feasibility and practicality of our approach.

Roadmap. Next, we present background (§ 2) with a brief overview of motivating studies and related works. § 3 analyzes and quantifies the benefits of optimizing the serving stack. § 4 explores the design choice we adopted for Configtron. § 5 presents our prototype and § 6 expands on the design challenges we faced and deployment scenario in production environment. Finally, § 7 provides concluding remarks.

2 Background and Related Works

We broadly define the CDN servers of a online service provider's network (Figure 1) as the user-facing servers that end-users interact with. Specifically, the content provider servers in points-of-presence (PoPs) and content distribution networks (CDNs).

The CDN's network stack, Figure 2, consists of the TCP protocol implementations, the web server (proxy) application, and content provider's web application (e.g. PHP or Java code). Traditional, these CDN servers employ broadly two different network stacks – one for user facing connections and another for the data center facing connections. The data center facing connections are often optimized to fully utilize the backhaul links. Whereas, the user facing connections are often fine-tuned to provide optimal performance in aggregate rather than to provide optimal performance per-client. To account for differences between a user's local conditions, the edge often deploys special web applications that infer properties of the user's local conditions and adjust multimedia images, html, and javascript to optimize. Essentially, the predominant approach is to reconfigure the CDN stack at the content layer, e.g., upon detecting a mobile client, many content providers redirect users to the mobile site.

This paper explores the benefits of extending reconfiguration from the web application down to the server application and the TCP/IP stack – and presents a system for realizing reconfiguration in a principled manner. In particular, there is a large space of potential parameters that can be explored from the application to the TCP/IP. To demonstrate the richness of the CDN server's parameter space, in Table 1, we present a representative list of these parameters and in § 3, we explore the impact of reconfiguring a subset of these parameters (Table 2).

2.1 Motivating Measurement Studies

Next, we discuss measurement studies that motivate the need for heterogeneity within the CDN's network stack:

Heterogeneity at the congestion layer (TCP): Although *Cubic* is used as the default congestion control algorithm, the Linux kernel includes over 10 variants of TCP. Moreover various measurement studies show that different variants are optimal for different networking conditions [27, 10]. Orthogonally, others [6, 2] have explored the impact of varying configuration parameters.

More recently, Google adopted UDP over TCP in their design of QUIC (Quick UDP Internet Connections) [14], thereby introducing more diversity into the stack. Existing studies of QUIC [20] show that QUIC outperforms TCP in mobile networks with high RTTs but performs sub-optimally (to TCP) in high bandwidth networks.

Layer	Configuration Parameters
Network (IP)	IP version, peering link.
Transport (TCP)	initial retransmission timeout (initRTO), autocork, congestion avoidance algorithm, send buffer size (wmem), initial congestion window (initcwnd), window scaling [19], receive buffer size(rmem), Nagle's Algo, TCP_low_latency.
Session (TLS)	TLS version, OSCP stapling, TLS false start, dynamic record sizing.
Application (HTTP)	HTTP version, push algorithm (HTTP/2), pipelining configuration (HTTP/1.1).
Content	video/img resolution, caching, compression algorithm.

Table 1: Representative list of configuration parameters.

Heterogeneity at the App layer (HTTP): SPDY [21], now HTTP/2, is used uniformly by Google and others over HTTP/1.1. However, recent studies [31, 8, 7, 20] provide strong empirical evidence that HTTP/1.1 outperforms SPDY (HTTP/1) under high packet loss rates and complex web-page dependencies, where multiple TCP connections perform better than SPDY's single, multiplexed TCP connection [31].

2.2 Related Work

Finally, we discuss related work on cross layer optimizations, creating standard interfaces for webstack, and configuration management.

Cross-layer optimizations: The most closely related work [2] explores the impact of cross layer configuration optimization for a limited set of configuration parameters, i.e., initial congestion window, HTTP pipelining, Appropriate Byte Count, and autocorking. Configtron explores a broader set of parameters (see Table 2) and presents a system to systematically tune these parameters in real time. While Configtron explores the transport and application layers, other have examined making changes at the content layer [26]: changing compression algorithms. As part of future work, we intend to extend Configtron to the content layer. Yet, others have explored tuning lower layers of the stack [11, 18, 29, 17] for the networking and link layers of wireless and mobile networks.

Configuration Management: Configtron's configuration management interface is motivated by existing attempts to expose TCP parameters and standardize the management interface for configuring TCP [13, 16]. Configtron extends on these approaches by encompassing more parameters and extending the management interface beyond TCP and into the HTTP and content layers. Whereas orthogonal approaches [13] directly collect information from the network, Configtron passively infers the state of the network conditions. Existing approaches to managing server configurations, focus on ensuring correct functionality and detecting misconfiguration [3, 28]. These approaches can be used to help improve the manageability of Configtron and debug performance problems that arise while using Configtron.

Takeaway While various measurement studies demonstrate the need for heterogeneous configurations, today's internet employs a "one size fits all strategy" where one set of configurations, suitable for a subset of the population, is used for the entirety of the internet. We note that unlike these prior studies that explore a single protocol or configuration parameter, in § 3 we present a more holistic exploration across multiple protocols, parameters and layers of the protocol stack. Furthermore, unlike prior work [2] we present a concrete system (§ 4) to reconfigure and optimize the different protocols and discuss design challenges (§ 6).

3 Empirical Study

To understand and quantify the benefits of reconfiguring the networking stack, we conducted a large scale study of the impact of selecting the optimal configurations over the default configuration parameters across different network conditions and websites.

3.1 Experiment Setup

To ensure reproducible and precise experiments across the different configuration combinations, we leverage MahiMahi [23], a proven network emulator for running and re-running web page load experiments. MahiMahi allows us to eradicate the variations in page load time (PLT) that may arise due to unpredictability of varying network conditions. Moreover, MahiMahi includes tools, called shells, that enable us to systematically modify and control network conditions – specifically, loss, bandwidth, and RTTs. Each page is loaded five times and the mean PLT is computed after filtering outliers. In the future, we also aim to look at SpeedIndex [15].

In our experiments, we explore these dimensions:

- **Network conditions:** We explore the traditional network properties: loss, latency, and bandwidth. To control these network properties, we use the following linux tools: NetEM and TC. To ground our study, we adjust the network latency, bandwidth, and loss to reflect realistic network conditions from various regions and networking infrastructure [1]. Network conditions tested include bandwidth of {0.3, 1, 5}Mbps, loss rate of {0, 1, 2.5, 5}% and delay of {50, 150, 250, 500}ms.

- **Server Network stack:** Table 2 presents the list of network stack configurations explored. Column 3 represents the default configuration values for Linux's network stack [24]. To reconfigure the CDN's network stack, the Linux kernel provides a

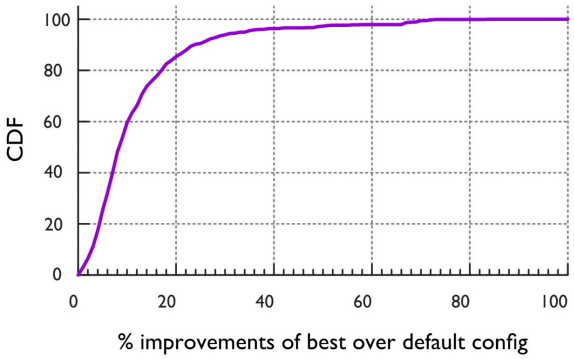

Figure 3: Benefits of using optimal configurations.

variety of options that can be tuned by changing kernel modules, using IOCTL socket calls, modifying IP tables or modifying application modules.

- **Website complexity:** We tested a variety of websites belonging to various categories; news, social networks, sports, business, e-commerce, and entertainment.

Layer	Protocol Parameters	Default	Values Tested
Transport (TCP)	tcp_congestion_control	Cubic	Reno, Cubic, Vegas, BBR
	initcwnd	10	3, 6, 9, 12, 15, 18
	tcp_slow_start_after_idle	1	0, 1
	tcp_low_latency	0	0, 1
	tcp_pacing	0	0, 1
	tcp_autocorking	1	0, 1

Table 2: CDN network stack configuration parameters.

3.2 Empirical Study on Reconfiguration

How inefficient is the "one-size-fits-all" strategy? We begin by exploring the implications of using sub-optimal configurations. In Figure 3 presents the difference in PLT between the default and the best configuration. We observe that in the median case, there is a 10% improvement in performance and in the tail (95th percentile) over a 40% improve in performance. We note that the tail conditions explored in our experiments are in fact representative of a large fraction of realistic connections (e.g., 2G connections in developing and emerging regions in Africa and Asia): Specifically, 68% of tail conditions have loss rates greater than 2.5% and bandwidth below 1Mbps. Moreover, over 40% of the gains are for content rich websites, e.g.,msn.com, tmall.com, espn.com and qq.com. We note that while these networks will eventually get updated, heterogeneity between different networks will always persist due to socioeconomic differences between regions.

Are some configurations strictly better than others? Figure 4 presents a comparison of the top four configurations from the experiments for *www.bbc.com*. The

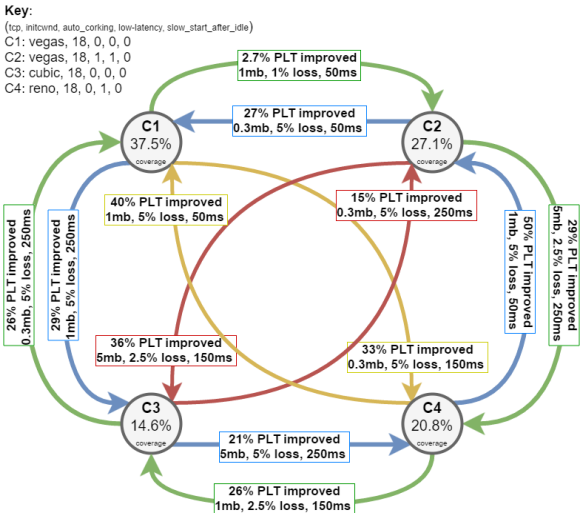

Figure 4: Comparison of top four optimal configurations.

coverage percentage (in circles) represents the percentage of conditions in which the given configuration works better than others, e.g., C1 is optimal for 37.5% of the conditions. Rectangular box show the PLT for the destination configuration over the source configuration for a specific network condition (*bandwidth, loss rate, delay*), e.g., C1 is 2.7% better than C2 for condition (1Mb, 1% loss, 50ms). Figure 4 shows that no configuration is strictly better than others — there is at-least one condition were each configuration is better or worst than the other configurations. Moreover the differences are staggering.

Is it easy to learn the best configuration for a specific network property? We start simple to answer the question of learning the best configurations by building C4.5 decision trees using the data from our experiments: the leaves of the tree are configuration parameters in Table 2 and the nodes are the predictive network conditions. To build the decision tree, we binned the different network conditions based on values. In Figure 5, we present decision tree for *www.youtube.com* — a representative decision tree. We summarized the decision tree and pruned nodes for more predictable configuration parameters: *slow_start_after_idle*, *low_latency* and *autocorking*. Our decision trees demonstrate that it is possible to learn the mapping of "optimal configuration" to network conditions.

4 Architecture

In Figure 6, we present the architecture for Configtron, our framework for proactively supporting the reconfiguration of CDN's network stack at large scale. We illustrate the functionality of the different components by

exploring the life cycle of a request.

When a new request arrives, the request router (our Layer 4 load balancer) sends it to a front-end server. Unlike traditional, Layer 4 load balancers, the request router contains meta-data mapping IP-prefixes to pools of VMs containing the appropriate configuration (for the specific IP-pool). For IP addresses that the request router contains no mappings of, the request router uses the default load balancing rules that sends the request to a pool of "default" servers — which are servers with default configuration. Requests for IP addresses with known configurations are directed to the appropriate pool of servers with those conditions. This pool of servers consists of a farm of appropriately configured VMs. In Configtron, the VM represents the granularity of reconfiguration — the level with which Configtron is able to configure (and reconfigure) the CDN's network stack. Moreover, configurations are done once, at the beginning of the connection.

Configuration Manager: The configuration manager generates a mapping of IP-addresses to optimal configuration parameters. To ensure scalability, the IP-addresses are aggregated by prefixes and the configuration parameters are aggregated to N different templates (where N is empirically derived using methodology described in Section 3). Aggregation along both directions minimizes the state maintained at the configuration manager and minimizes fragmentation of resources while incurring a slight performance inefficiency.

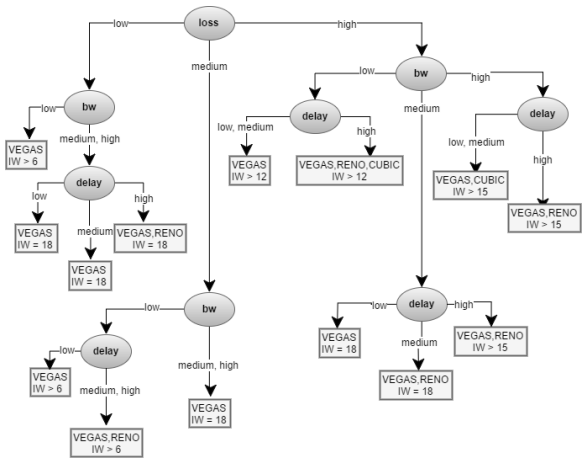

Figure 5: Decision tree for youtube.com.

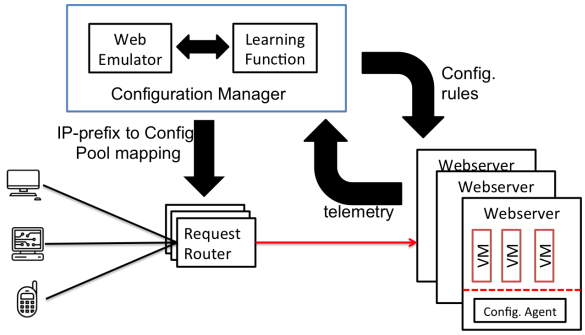

Figure 6: System Architecture

Additionally, the configuration manager leverages a realistic emulator to test out different configuration values with different networking conditions and use a learning function to learn the optimal configuration. The exact details of the learning function are beyond the scope of this work and we merely sketch out the functional requirements. We expect that the learning function can be implemented using a variety of machine learning techniques, e.g., deep reinforcement learning or decision trees. Abstractly, the learning function takes as input the inferred (measured) RTT, loss rate, bandwidth, and website structure, then explores different configuration parameters, and selects the parameters that optimize web page load times.

Finally, the configuration manager maintains a constant pool of "free servers" each configured with the "N"-golden templates. This pool of "free servers" ensures that new requests do not have to be delayed waiting for a new server to be configured.

Config Agent: This runs within the hypervisor of the different physical servers, instantiating VMs with prespecified configurations. The agent provides the configuration manager with a standard and a uniform interface across different servers regardless of the OS (Linux, Windows) and the web-server application (Apache, NGinx). Moreover, the agent collects statistics for each connection (IP address) including the RTTs, loss, bandwidth, and jitter.

5 Prototype Implementation

To explore the feasibility and viability of Configtron, we have developed an initial prototype and are exploring the implications of employing it on a cluster of 300 servers. Our prototype implementation is based on the discussion in § 4.

Prototype Implementation. We have implemented the Config agent in Python in 890 lines of code. The Config agent provides controls over the configuration parameters discussed in § 3. It runs locally on each backend VM and gets instructions from configuration manager for modifying TCP and HTTP parameters. The Configuration manager uses MahiMahi for the web emulator and uses a simplicity learning function based on a decision tree. Our current prototype provides fine-grained control. The request router is implemented as an SDN switch which enables both physical and virtual deployments.

6 Design Challenges and Future Works

Our design and prototype explore a single point in the design space. In this section, we discuss alternate design points and their implications.

Inferring Network Conditions: Our current design infers the client's network conditions based on packets exchanged between servers and the clients; in a similar manner to how TCP learns a client's network conditions. An alternate and more direct approach is to have the clients explicitly probe, capture and exchange network condition information with the servers. This can be done by modifying the client software stack [32] to actively inform the CDN server or adding javascript or invisible images into the webpage that enables the webserver to collect client-side statistics [12]. As part of future work, we plan to explore the more accepted approach: embedding javascript or images for performance profiling.

Learning Configurations: Online Versus Offline Section 3 demonstrates that we can build decision trees and learn optimal configurations for different network conditions, however, these requires brute force and extensive testing which may be infeasible when we explore the broader space of configuration parameters and explore the more nuanced network conditions that appear in practice. We plan to explore the use of online learning techniques, e.g., A-B testing [30], and compare the effectiveness of online techniques with offline techniques.

Configuration Granularity: Our current design reconfigures the webstack at the granularity of a VM — with each VM containing a distinct set of configuration parameters. While heavy weight, VM-level configurations allows us to reuse existing and mature tools while providing full control over all configuration parameters. Alternatively, we could explore containers (e.g., LXC or Docker) as the granularity of control. However, since containers share the same network stack, we would need to use userspace TCP/IP protocols to provide control over certain TCP configurations, e.g., TCP version. As part of future work, we plan to explore a fusion of both extremes. Namely, leverage VM-level granularity for controlling kernel-level global parameters, e.g., TCP version, and container-level granularity for controlling connection-level local parameters, e.g., HTTP protocol version.

Reconfiguration Frequency: Our current design reconfigures the CDN network stack at the beginning of each connection. This limits the flexibly of Configtron and prevents us from adapting to drastic changes in the end-user's networking conditions. Alternatively, we could reconfigure the stack at finer granularities, i.e., before every packet or before every object. Although reconfiguration at the finer granularity enables us to react more finely, and ultimately to improve performance, finer granularity introduces several significant challenges, e.g., managing reconfiguration overheads and tackling the implications of reconfiguration on existing state for the connection.

Reconfiguration Overheads: There are some overheads associated with reconfiguring the CDN network stack, specifically, VM setup cost (e.g., image transfer and bootup) and kernel reconfiguration (e.g., changing the protocol version). To tackle these overheads, we plan to explore a combination of approaches to ameliorate this overheads, e.g., maintaining a fleet of preconfigured VMs and proactively scale-up this fleet in respond to fluctuations in demand.

Deployment Scenario: The current design explores a point in the design space that requires content-providers to modify and improve their infrastructure. Yet, there are also points in the design space; where the content-provider and the end-user cooperate, these points in the design space enable us to explore a broader range of configuration options including configuration on the end-user client and explicitly exchange of client side information. Although this approach appears altruistic, this point in the design space can be easily explored by large online service providers, e.g., Facebook and Google. As part of future work, we plan to explore the additional benefits that arise from leveraging client (end-user) cooperation.

7 Conclusion

In this paper, we argue that content providers should eschew the "one-size-fits-all" approach to configuring CDN network stacks and instead embrace heterogeneity in CDN network stack configurations. To support our argument, we perform an empirical evaluation of the implication of configuration and find that heterogeneity can lead to significant improvements.

This paper takes the first step towards realizing heterogeneity by proposing an open but simple interface for configuring the network serving stack and introducing a framework that enables a CDN to practically leverage heterogeneity. Our framework learns network conditions and enables the use of machine learning techniques to determine the optimal configuration for the different network conditions.

8 Discussion

In this paper, we proposed that "one-size-fits-all" approach to tuning/configuring server networking stacks result in sub-par performance for some end-users, especially those users in emerging regions. Due to the ever-expanding nature of internet, all end-users do not face similar network conditions and improvement in underlying protocols do not uniformly benefit all users [31, 8]. This argument stands in stark contrast to the traditional setup of server networking stacks where a single network configuration is used for a divergent set of users.

Expected Feedback Our proposal for dynamic reconfiguration of the CDN network stack is grounded on emulations and prototype implementations. We are looking for feedback on challenges that can arise when deployed in large scale, production environment.

- **Management Overheads:** Dynamically reconfiguring the CDN protocol stack complicates performance diagnosis and troubleshoot. We plan to investigate methods for reducing this complexity, e.g., minimizing the number of active configuration combinations.
- **QuiC:** Google employs QuiC, which utilizes UDP and not TCP. Yet, QuiC has a number of configuration options thus making the underlying principles of Configtron immediately applicable to QuiC.
- **Long-lived Connections:** Configtron configures the CDN stack at the beginning of the connection and this prevents us from dealing with drastic changes in the network which require reconfiguring existing connections. Fortunately, TCP's congestion control algorithm is designed to explicitly handle these dynamic situations. Configtron attacks an orthogonal problem and focuses on improving TCP (and other protocols) by tuning the configuration of their internal algorithms.
- **Broader Evaluations and QoE Metrics:** As part of ongoing work, we are planning to understand the limits of Configtron by evaluating Configtron across a larger parameter space; a wide range of network conditions (e.g., mobile networks or bufferbloat) and dynamics (e.g., time of day effects); and a broader set of web page QoE metrics (e.g. SpeedIndex) and Video QoE metrics.

References

[1] Akamai. Akamai state of the internet. `https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report`.

[2] Mohammad Al-Fares, Khaled Elmeleegy, Benjamin Reed, and Igor Gashinsky. Overclocking the yahoo!: Cdn for faster web page loads. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 569–584. ACM, 2011.

[3] Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1–11, 2010.

[4] Michael Butkiewicz, Harsha V Madhyastha, and Vyas Sekar. Understanding website complexity: measurements, metrics, and implications. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 313–328. ACM, 2011.

[5] Cisco. White paper: Cisco vni forecast and methodology, 2015-2020. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html`.

[6] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing tcp's initial congestion window. *Computer Communication Review*, 40(3):26–33, 2010.

[7] Yehia Elkhatib, Gareth Tyson, and Michael Welzl. Can spdy really make the web faster? In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

[8] Jeffrey Erman, Vijay Gopalakrishnan, Rittwik Jana, and Kadangode K Ramakrishnan. Towards a spdyier mobile web? *IEEE/ACM Transactions on Networking*, 23(6):2010–2023, 2015.

[9] Facebook. Building for emerging markets: The story behind 2g tuesdays. `https://code.facebook.com/posts/1556407321275493/building-for-emerging-markets-the-story-behind-2g-tu`

[10] Sally Floyd. Highspeed tcp for large congestion windows. 2003.

[11] Majid Ghaderi, Ashwin Sridharan, Hui Zang, Don Towsley, and Rene Cruz. Tcp-aware resource allocation in cdma networks. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 215–226. ACM, 2006.

[12] Mojgan Ghasemi, Partha Kanuparthy, Ahmed Mansy, Theophilus Benson, and Jennifer Rexford. Performance characterization of a commercial video streaming service. In *Proceedings of the*

2016 ACM on Internet Measurement Conference, pages 499–511. ACM, 2016.

[13] Monia Ghobadi, Soheil Hassas Yeganeh, and Yashar Ganjali. Rethinking end-to-end congestion control in software-defined networks. In *Proceedings of the 11th ACM Workshop on Hot Topics in networks*, pages 61–66. ACM, 2012.

[14] Google. Quic, a multiplexed stream transport over udp. `https://www.chromium.org/quic`.

[15] Google. Speed index. `https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index`.

[16] Jens Heuschkel, Michael Stein, Lin Wang, and Max Mühlhäuser. Beyond the core: Enabling software-defined control at the network edge. In *Networked Systems (NetSys), 2017 International Conference on*, pages 1–6. IEEE, 2017.

[17] Fan Li and Guizhong Liu. Cross-layer optimization for multiuser video streaming over wireless networks. 2008.

[18] Xiaojun Lin, Ness B Shroff, and Rayadurgam Srikant. A tutorial on cross-layer optimization in wireless networks. *IEEE Journal on Selected areas in Communications*, 24(8):1452–1463, 2006.

[19] Linux Programmer's Manual. tcp - tcp protocol. `http://man7.org/linux/man-pages/man7/tcp.7.html`.

[20] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. How quick is quic? In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[21] Roberto Peon Mike Belshe. Spdy protocol. `https://tools.ietf.org/id/draft-mbelshe-httpbis-spdy-00.txt`.

[22] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. Polaris: Faster page loads using fine-grained dependency tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, 2016.

[23] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *USENIX Annual Technical Conference*, pages 417–429, 2015.

[24] Linux programmer's manual. Tcp protocol man page. `http://man7.org/linux/man-pages/man7/tcp.7.html`.

[25] Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, Oliver Spatscheck, and Walter Willinger. Tcp revisited: a fresh look at tcp in the wild. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 76–89. ACM, 2009.

[26] Shailendra Singh, Harsha V Madhyastha, Srikanth V Krishnamurthy, and Ramesh Govindan. Flexiweb: Network-aware compaction for accelerating mobile web transfers. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 604–616. ACM, 2015.

[27] Kun Tan Jingmin Song, Q Zhang, and M Sridharan. Compound tcp: A scalable and tcp-friendly congestion control for high-speed networks. *Proceedings of PFLDnet 2006*, 2006.

[28] Ya-Yunn Su, Mona Attariyan, and Jason Flinn. Autobash: improving configuration management with operating system causality analysis. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 237–250. ACM, 2007.

[29] Srisakul Thakolsri, Wolfgang Kellerer, and Eckehard Steinbach. Qoe-based rate adaptation scheme selection for resource-constrained wireless video transmission. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 783–786. ACM, 2010.

[30] Kaushik Veeraraghavan, Justin Meza, David Chou, Wonho Kim, Sonia Margulis, Scott Michelson, Rajesh Nishtala, Daniel Obenshain, Dmitri Perelman, and Yee Jiun Song. Kraken: leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, pages 635–650. USENIX Association, 2016.

[31] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. How speedy is spdy? In *NSDI*, pages 387–399, 2014.

[32] Yiannis Yiakoumis, Sachin Katti, and Nick McKeown. Neutral net neutrality. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 483–496. ACM, 2016.