A Cloud-based Content *Gathering* Network

Debopam Bhattacherjee
ETH Zürich

Muhammad Tirmazi
LUMS

Ankit Singla
ETH Zürich

Abstract

Many popular Web services use CDNs to host their content closer to users and thus improve page load times. While this model's success is beyond question, it has its limits: for users with poor last-mile latency even to a nearby CDN node, the many RTTs needed to fetch a Web page add up to large delays. Thus, in this work, we explore a complementary model of speeding up Web page delivery – a content gathering network (CGN), whereby users establish their own geo-distributed presence, and use these points of presence to proxy content for them. We show that deploying only 14 public cloud-based CGN nodes puts the closest node within a median RTT of merely 4.8 ms (7.2 ms) from servers hosting the top 10k (100k) most popular Web sites. The CGN node nearest to a server can thus obtain content from it rapidly, and then transmit it to the client over fewer (limited by available bandwidth) high-latency interactions using aggressive transport protocols. This simple approach reduces the median page load time across 100 popular Web sites by as much as 53%, and can be deployed immediately without depending on any changes to Web servers at an estimated cost of under $1 per month per user.

1 Introduction

Response times for Web services have a significant bearing on user experience, and hence revenue for service providers. For Google, a few hundred milliseconds of increase in latency translate into a significant reduction in the number of searches per user [9]. Similar figures have been reported by other providers, including Bing [11] and Amazon [17]. There is thus significant interest in improving latency in Web page delivery.

The predominant model for Web page delivery today is the use of content distribution networks, such as Akamai, Limelight, and Cloudflare, and in the case of the largest Web service providers, their own globally distributed infrastructure. By establishing an extensive global presence, CDNs attempt to locate their customers' content as close to their users as possible, thus reducing latency. For instance, Akamai leverages their 216,000+ servers located in more than 1500 networks around the World, to be able to claim [2]:

Eighty-five percent of the world's Internet users are within a single "network hop" of an Akamai CDN server.

It is beyond question that this model yields significant performance improvements across large numbers of users. However, we observe that for clients with poor last-mile latency, the benefit from CDNs is much more limited – if the last mile itself adds, say, 100 milliseconds of latency, then the many RTTs incurred in a typical Web page request [8] will still add up to a large page load time. It is worth noting that such large last-mile latencies are not atypical in many parts of the World [6]. There is, of course, significant ongoing work both on cutting down the number of RTTs a Web request takes, and on improving last-mile latencies. However, realizing these benefits across the large eco-system of Web service providers and ISPs could still take many years[1]. How do we improve performance for users in these environments today?

To address this problem, we explore a starkly different model for Web content delivery than today's CDN model. Instead of bringing content closer to the users, we aim to push users closer to the content. We observe that present public cloud infrastructure provides enough global points of presence to be able to reach within close proximity of most popular Web services, and thus provides a natural platform to build a "Content Gathering Network", which operates on behalf of users. A user then always has a **CGN** point-of-presence close to any desired Web service. Such a CGN node, due to its low-latency connectivity to the Web service, can fetch content quickly, even if it involves many (extremely short) RTTs, and then ship it to the client over few (ideally, one in case of infinite available bandwidth) high latency interactions, using aggressive transport protocols. We show that this approach can reduce page load times by 53% in the median for popular Web sites from a client in Lahore, Pakistan. Further, this approach is immediately deployable at a surprisingly low cost — under $1 per user per month.

Beyond the objective of improving performance for clients with poor last-mile latency, we believe this preliminary work raises fundamental questions regarding the present model of Web content delivery – can we leverage public cloud infrastructure and the observation that most content is hosted in (or near) this infrastructure to entirely eliminate CDNs from Web site delivery? Note that we deliberately scope the question narrowly for Web site delivery, where the number of bytes delivered is

[1] Optimizations like WebP and SPDY, which have been available for many years, have adoption rates under 1% across popular Web sites [1].

small relative to the content volume on the Internet – for applications like video streaming, it is clear that CDNs also help in substantially cutting the need for wide-area bandwidth.

2 Related work

Reducing latency in Web content delivery is a hot topic with research efforts along many directions.

Cloud-assisted browsers: Various split-architecture browsers have been proposed to reduce latency. Opera's Mini/Turbo [20, 19] aims to reduce latency by compressing Web content at a proxy before delivering it to users. Google's Flywheel [1] is a similar proxy service for Chrome browser. Amazon Silk [5] is a split-architecture browser which offloads processing from thin clients to well-provisioned servers in the cloud which analyze traffic patterns, preprocess content and apply machine learning algorithms in order to reduce latency. These efforts focus on either content compression, or processing-offload to preserve bandwidth and compute resources at clients, but their impact on latency is not consistently positive, as Sivakumar *et al.* [27] show in their analysis of a popular cloud-assisted browser. Further, benefits from processing-offload and compression are both orthogonal to our work, and could yield substantial further benefit beyond what we evaluate here.

WebPro [25] maintains a database of resource lists for popular Web sites at the proxy, proactively fetching related resources on receiving client requests and sending bulk responses back to the clients. PARCEL [28] packages several optimizations including compression, processing-offload, and batching of data to reduce the number of RTTs to reduce load times for mobile devices suggesting the use of a proxy "implemented on a powerful server". Similarly, Cumulus [18], uses "a well-provisioned cloud server" to fetch content from Web servers and send bulk responses to the clients to reduce the number of round trips. However, this past work does not exploit our observation on the proximity of cloud nodes to popular Web services to propose a distributed content gathering network to further reduce latency.

Measurement studies: Prior work [15] evaluated the use of cloud services by popular Web sites, finding that 4% of Alexa's top 1 million domains use Amazon EC2 or Microsoft Azure. Our results quantify *latency* to the Web servers directly via measurements, as opposed to looking at IP prefixes etc. to identify where the servers are hosted. It is also possible that the consolidation of Web services in (or near) cloud infrastructure has grown in the 4+ year period between those measurements and ours. Our finding of the 100,000 most popular Web sites being a mere 7.2 ms away from EC2 nodes in the median, is the key underpinning of our suggested approach.

Protocol enhancements: While a large number of protocol and content-encoding enhancements are being fleshed out in the community, even those with the backing of industry giants like Google (SPDY [12], WebP [14]) have seen extremely low adoption rates of under 1% across Web servers [1]. Our work provides an easy vector to improve Web performance *without* depending on Web servers. In fact, it effectively serves as a vehicle for deploying protocol enhancements – the close proximity of our CGN nodes to Web servers implies that deploying certain enhancements on CGN nodes could be almost as effective as deploying them to Web servers. We ourselves leverage known techniques such as larger TCP window sizes [10] and persistent connections via mechanisms like TCP cookies [22].

3 Cloud consolidation of Web services

The motivating observation behind this work is that increasingly, popular Web services are hosted in, or near, a small set of public cloud data centers. We first quantify the extent of this "cloud consolidation".

We deployed one node in each of Amazon's 14 data center regions. From each of these 14 nodes, we measured round trip times to each Web server hosting the top 100,000 Web sites in Alexa's list of popular sites [3]. We used `hping` [24] to conduct our RTT measurements, allowing us to send TCP SYN packets to the Web servers and record when the TCP SYN-ACKs were received at our Amazon nodes. For a smaller set of Web servers (top 10,000), we also similarly measured RTTs from clients in Lahore, Pakistan and Zürich, Switzerland. Both of these clients are university-hosted and "real" end-user connectivity is likely to be worse. An investigation using large scale experiments across more clients on the RIPE Atlas platform [23] is underway.

Note that we filter out Web servers in China from our measurements. Latencies to Chinese servers from our 14 EC2 locations are higher, but only due to bureaucratic hurdles: Amazon does in fact provide an EC2 region in China's high population Beijing area, but using this region requires registering a Chinese legal entity [7].

3.1 Most servers are near an EC2 node

Fig. 1 shows the CDF of RTTs to different Web servers from our 14 Amazon EC2 nodes, as well as our clients in Lahore and Zürich. The EC2 data shown is for the EC2 node which gave the *lowest* RTT to the Web server. In line with our expectations, latency is lowest from EC2 and highest from Lahore, with the medians (90^{th}-percentiles) being 4.8 ms (39.5 ms), 39.8 ms (215.8 ms), and 275.8 ms (471.8 ms) from EC2, Zürich, and Lahore respectively. Thus, median RTT from EC2 is 8× smaller than from Zürich, and 57× smaller than from Lahore.

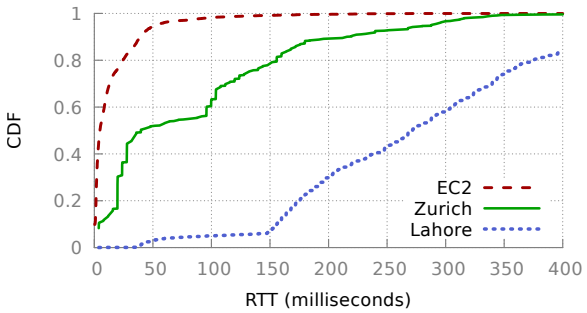

Figure 1: *CDF of RTTs to the top* 10,000 *popular Web servers from Lahore, Zürich and our* 14 *EC2 nodes.*

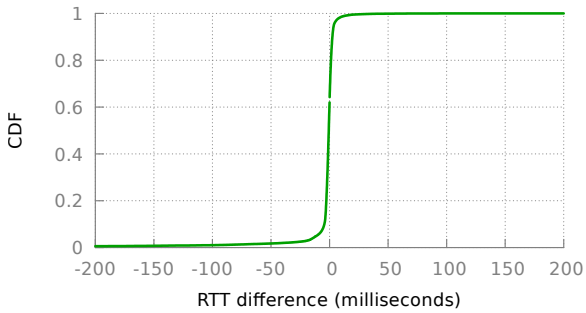

Figure 2: *CDF of differences in RTTs to the top* 10,000 *popular Web servers from their closest EC2 node after one week.*

For the top 100,000 domains (Fig. 1 shows results for the top 10,000), the median RTT from EC2 was similarly small — 7.2 ms, indicating that this consolidation in or near EC2 continues across less popular Web sites, although to a somewhat lesser extent.

An interesting feature in Fig. 1 is the emergence of a few significant "steps" in the measured RTTs from Zürich – nearly 14% of the mass is in one such step between 19.7-20 ms. We believe this corresponds to a set of Web servers located in precisely the same data center, which is ∼20 milliseconds away from Zürich. (For most of these Web servers, the EC2 node in Frankfurt is within 10 milliseconds RTT.) Several other such steps can be seen clearly in the plot. In ongoing work, we are investigating more thoroughly, a mapping between these steps and different cloud data centers.

From Lahore, most Web servers are far: 94% being more than 150 ms away. We conjecture that the step-characteristic of the Zürich measurements is absent here due to greater latency variations across longer paths.

There is also a "plateau" in the plot between 50-100 ms in the measurements from Zürich, where there are few measurements. This is due to the trans-Atlantic latency from Zürich to servers in the Americas.

These results show that most Web services are hosted either within the same data center as their closest EC2 nodes, or some other data center in the same regions, or in some infrastructure with good connectivity to our EC2 nodes. It is likely that we could further decrease the RTTs from our cloud nodes by including nodes from other cloud providers, such as Microsoft Azure, or even different Amazon data centers in the same regions.

3.2 Are these observations stable?

To be able to exploit the above observation, we also need to ascertain whether these measurements stay stable – at the least, does the EC2 location which was closest to a Web server show similar RTT to the Web server across measurements separated by long time intervals? To answer this question, we created a mapping of each Web server to its closest EC2 node in the above experiments. After a period of one week from the initial measurements, we used the same methodology to measure the RTTs between Web servers and the EC2 nodes they are mapped to. Fig. 2 shows the CDF of changes in RTT across server-EC2-node-pairs between these measurements. The change in latencies is extremely small, the median change being 0.6 ms, with 90% of the changes lying within a 15 ms range around zero. Thus, the EC2 nodes mapped to Web servers continue to provide low latency access to them across time. Even if a mapped EC2 node is no longer the *nearest* one to a Web server, the change in RTT is small.

4 A Content Gathering Network

We propose to exploit the proximity of cloud-hosted nodes to most Web servers to build a content gathering network (CGN), so that users are able to get Web content faster. The intuition behind our approach is illustrated in Fig. 3. In a typical Web request, there are a large number of round-trips between the client and the server. If the last-mile latency at the client is poor, even round-trips with a geographically nearby CDN node (instead of the authoritative content server) can incur high latency. However, if a client can identify a **CGN** node near the Web server and proxy its request through it, the CGN node can obtain content from the server over a small network RTT, before delivering it to the client. The client and the CGN node use aggressive transport protocols, with large initial window sizes, and without the need of an additional handshake (unlike TCP). These are standard techniques in networking [22, 10].

Our design requires CGN nodes to maintain a mapping of Web servers to CGN nodes. Each CGN node makes independent latency measurements and then shares measurement digests (few MB even for a million popular Web sites) with other CGN nodes, thus being able to compute the mapping. Clients then obtain the mapping

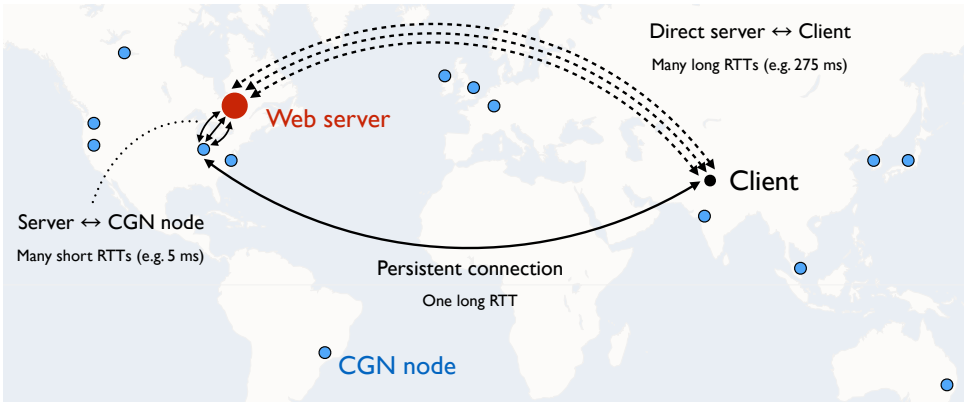

Figure 3: *There are many long RTTs between the client and the server in the default case. These are replaced by few (ideally, one when available bandwidth is infinite) long RTTs between the client and the CGN node, plus many short RTTs between the CGN node and the Web server.*

from CGN nodes. Given the stability of the mapping (§3.2), measurements, computations, transfers, and updates of mappings are all infrequent (*e.g.,* once a day).

A client's browser forwards Web requests to a local HTTP proxy service, which then forwards the URL to the CGN node nearest to the hosting Web server. The CGN node runs a headless browser, which starts loading the Web page. As the content is downloaded, the CGN node starts to forward it to the client's local proxy in parallel using a TCP byte stream. The local proxy, in turn, serves the content to the browser, which starts loading the page.

Ideally, delivering the content from the CGN to the client should be bounded only by bandwidth-based constraints and one RTT. Given that serving Web requests often involves significant interaction between the CDN front-ends and back-end or origin servers [21], some long RTTs are often incurred even in the CDN model. Thus, if CGN achieves 1 RTT page load times (modulo bandwidth constraints), then perhaps the role of CDNs in Web page delivery can be substantially reduced, or even entirely eliminated. However, our present implementation often does not hit this goal due to the difference in behaviors between client-side browsers and headless browsers at the CGN. Due to this, to prevent many long RTTs to the CGN, we still rely on traditional CDN nodes to serve content from these secondary requests when we can identify the use of a CDN through simple domain name filters[2]. But we do not believe that the 1-RTT goal is unreachable, and are working towards it.

5 Evaluation

We use `m4.10xlarge` EC2 instances (40 vCPUs, 160 GB memory, and 10 Gbps bandwidth) as CGN nodes, and *PhantomJS* as the headless browser. Presently, for our experiments, we only operate 2 CGN nodes (North California, USA and Frankfurt, Germany) and test only those domains which would have mapped to either of these 2 locations; an actual deployment would use all 14 nodes (and possibly more at other cloud providers) to serve requests at each location. At the client, we use the Chrome browser. For automating Web page loads and recording page load times (PLTs), we used `sitespeed.io` [26].

5.1 Improvement in PLT

We loaded Web pages with and without the CGN from a client in Lahore, reflecting our high last-mile latency scenario. Nevertheless, as noted in §4, content requests to CDNs directly go to the CDN servers while all other requests are served through our CGN. Out of the Alexa top-10k, we evaluate the top 100 and random 100 Web pages which map to either of our two CGN nodes. For the top-100 set, as Fig. 4 shows, the median page load time reduces from (the default) 16.1 seconds to 7.6 seconds with CGN — a reduction of 53%. For the random-100 set (Fig. 5), median PLT is reduced by 43.2%.

5.2 Comparison with Flywheel

While a detailed comparison is left to future work, we present results from a small comparison with Google's Flywheel [1]. While Flywheel is targeted at mobile devices, the same functionality is also made available by Google for the desktop browser in the form of the "Data Saver" extension [13]. Unfortunately, our automation and measurement framework based on `sitespeed.io` does not work with Chrome extensions, and thus these experiments could not be fully automated yet, so we test a small set of 30 top domains with Flywheel, with CGN, and without either. As Fig. 6 shows, CGN achieves significant gains beyond Flywheel's, with median PLTs of

[2]Possibly, we also benefit to some extent from anycast-based CDNs; teasing out their impact on our results is left to future work.

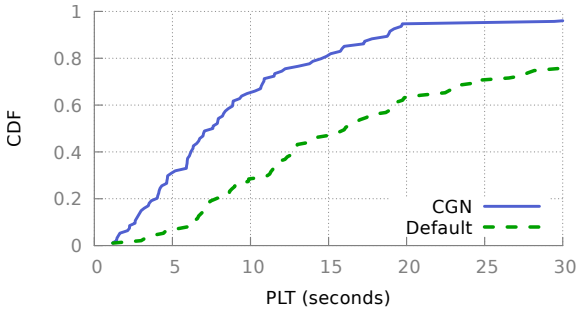

Figure 4: *PLTs from Lahore for the top* 100 *domains which map to North California or Frankfurt.*

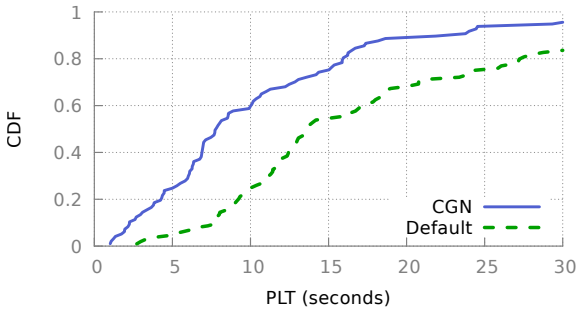

Figure 5: *PLTs from Lahore for* 100 *random domains which map to North California or Frankfurt.*

14.53 seconds, 12.37 seconds and 9.81 seconds when loaded without either, with Flywheel, and with CGN respectively. We note that Flywheel is a complex system with numerous optimizations, including compression, caching, and prefetching at the proxies. These techniques are all orthogonal to ours, and could be readily added to our approach for even more reduction in PLTs.

5.3 Isn't this very expensive?

At first, it may appear that operating such a globally distributed infrastructure on behalf of users could be exorbitantly expensive. However, we find that with multiplexing across potential users of such a service, this need not be the case. For an average request, we find that a CGN node uses ∼300 ms CPU time and ∼2 MB network bandwidth in both directions. Suppose that the average user requests 5,000 pages per month[3]. Conservatively using the highest costs for EC2 nodes[4] leads to a cost estimate of $0.934 per user per month. Even if the system operates at substantially lower efficiency, we believe the costs would be reasonable — for Lahore, they amount to ∼10% of the cost of typical broadband plans.

[3]Most users seem to browse substantially less than this [29].

[4]Reserved `m4.10xlarge` instances with upfront payment are priciest at Sao Paulo — $1.828/hour [4]. Bandwidth costs are $0.01/GB.

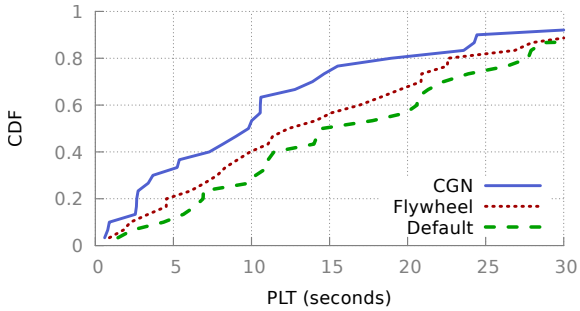

Figure 6: *PLTs from Lahore for the top* 30 *domains which map to NC or Frankfurt, with CGN, with Flywheel, and with neither.*

While we believe users stuck with poor connectivity would be willing to pay the minimal costs, this would perhaps be undesirable, in that it shifts responsibility from the content-provider and CDN eco-system onto users. Several alternative models could work: (a) competing cloud providers (Amazon, Azure, Rackspace) may bear the cost to attract more Web service providers to their infrastructure; (b) browser vendors competing for market share could run the service; (c) Web service providers themselves may incur the expense.

6 Conclusion

We present measurements showing the massive consolidation of Web servers in or near a small number of cloud data centers – Web servers hosting the top 100k domains are within a 7.2 ms RTT (median) from only 14 cloud nodes. Exploiting this consolidation, we propose a design for speeding up Web page delivery, especially for users behind a poor last-mile, showing that PLTs can be reduced by 43 – 53% following this simple approach, even with our preliminary implementation. We believe there are significant opportunities for further improvement, including better engineering at the CGN nodes to speed up the headless browser, prioritizing content we believe the client will need earlier in the page load, incorporating techniques like compression and caching developed for other systems like Flywheel, etc.

This work also raises interesting questions about whether a simpler model can effectively compete with the predominant CDN based model for Web page delivery. Our present implementation still uses CDNs for some content, but we are exploring ways of obtaining near-optimal Web page delivery in the typical case without the use of CDNs.

7 Discussion topics

The CGN model we propose is a significant departure from how Web page delivery works today. Apart from the "who pays" question addressed above, we invite discussion and feedback on several topics:

CGN v. CDN: We believe the CGN model presents different trade-offs than the CDN model – instead of CDNs operating hundreds of thousands of servers across thousands of locations, perhaps similar or even better performance can be achieved with a CGN. For entirely static content, if CDNs were to use similarly aggressive transport protocols, the latency difference would come down to the CGN-client latency v. CDN-client latency, and the latter would typically be smaller. However, for dynamic content, CDNs often need connectivity to the back-end or content-origin servers [21], ameliorating their client proximity advantage in this comparison. The substantially higher cost and complexity of CDNs thus may not be worth the expense (for low-traffic-volume applications like Web page delivery).

A matter of time? If Web servers and CDNs do deploy aggressive protocol enhancements, the CGN approach could lose its utility. However, as pointed out earlier (§2) even the most popular Web servers are slow to adopt enhancements, even when they are backed by industry giants. Further, servers are often conservative in their resource-use per user (to protect against DoS), while our design could potentially create more accountability at the user end, and can thus afford to be less conservative.

Security, HTTPS: We do not foresee significant challenges in supporting HTTPS, but the client browser needs to trust the CGN instead of treating it as an eavesdropper, because the CGN node would need to act as a TLS/SSL proxy. Note than CDNs already operate under a similar trust model. There is also scope for using techniques like Intel SGX [16] to hide the *content* of client-server interactions from the CGN nodes as a privacy enhancement, although CGN nodes will still be able to see *which servers* a client connects to. CGN nodes would also need effective sandboxing of user requests from each other. We note that CDNs also provide security services like DDoS protection. However, there is no reason cloud infrastructure cannot do the same.

Fairness: The use of aggressive transport protocols raises issues of fairness to Internet hosts not using them. However, short flows like those for Web page delivery are inherently disadvantaged in comparison to longer-lived traffic for applications like video streaming. Further, Web browsers already break flow-fairness by employing multiple parallel TCP connections. Nevertheless, we plan to study the fairness-performance trade-off in greater depth in future work.

References

[1] Agababov, V., Buettner, M., Chudnovsky, V., Cogan, M., Greenstein, B., McDaniel, S., Piatek, M., Scott, C., Welsh, M., and Yin, B. Flywheel: Google's Data Compression Proxy for the Mobile Web. *Usenix NSDI* (2015).

[2] Akamai. Facts & Figures. https://www.akamai.com/us/en/about/facts-figures.jsp. [Online; accessed 14-March-2017].

[3] Amazon Web Services, Inc. Alexa Top Sites. https://aws.amazon.com/alexa-top-sites/. [Online; accessed 07-March-2017].

[4] Amazon Web Services, Inc. Amazon EC2 Reserved Instances Pricing. https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/. [Online; accessed 14-March-2017].

[5] Amazon Web Services, Inc. Amazon Silk Documentation. https://aws.amazon.com/documentation/silk. [Online; accessed 06-March-2017].

[6] Awan, M. F., Ahmad, T., Qaisar, S., Feamster, N., and Sundaresan, S. Measuring broadband access network performance in Pakistan: A comparative study. *IEEE Conference on Local Computer Networks* (2015).

[7] AWS (China). AWS (China) FAQs. https://www.amazonaws.cn/en/about-aws/china/faqs/. [Online; accessed 14-March-2017].

[8] Bozkurt, I. N., Chandrasekaran, B., Aguirre, A., Godfrey, P. B., Laughlin, G., Maggs, B., and Singla, A. Why is the Internet so slow? *Passive and Active Measurement Conference* (2017).

[9] Brutlag, J. Speed matters for Google web search. http://goo.gl/vJq1lx, 2009. [Online; accessed 06-March-2017].

[10] 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. *ACM CCR* (2010).

[11] Eric Schurman (Bing) and Jake Brutlag (Google). Performance Related Changes and their User Impact. http://goo.gl/hAUENq. [Online; accessed 14-March-2017].

[12] Erman, J., Gopalakrishnan, V., Jana, R., and Ramakrishnan, K. K. Towards a SPDYier mobile web? *IEEE/ACM Transactions on Networking* (2015).

[13] Google. Data Saver. https://goo.gl/JBKCDz. [Online; accessed 07-March-2017].

[14] Google Developers. WebP: A New Image Format For The Web. http://developers.google.com/speed/webp/. [Online; accessed 07-March-2017].

[15] He, K., Fisher, A., Wang, L., Gember, A., Akella, A., and Ristenpart, T. Next stop, the cloud: Understanding modern web service deployment in ec2 and azure. *ACM IMC* (2013).

[16] Intel developer Zone. Intel Software Guard Extensions (Intel SGX). https://software.intel.com/en-us/sgx. [Online; accessed 14-March-2017].

[17] Liddle, J. Amazon Found Every 100ms of Latency Cost Them 1% in Sales. http://goo.gl/BUJgV. [Online; accessed 14-March-2017].

[18] Netravali, R., Sivaraman, A., Das, S., Goyal, A., Winstein, K., Mickens, J., and Balakrishnan, H. Mahimahi: Accurate Record-and-Replay for HTTP. *USENIX ATC* (2015).

[19] Opera Software. Data savings and turbo mode. http://www.opera.com/turbo. [Online; accessed 06-March-2017].

[20] OPERA SOFTWARE. Opera Mini. http://www.opera.com/mobile/mini. [Online; accessed 06-March-2017].

[21] PUJOL, E., RICHTER, P., CHANDRASEKARAN, B., SMARAGDAKIS, G., FELDMANN, A., MAGGS, B. M., AND NG, K.-C. Back-office web traffic on the internet. *ACM IMC* (2014).

[22] RADHAKRISHNAN, S., CHENG, Y., CHU, J., JAIN, A., AND RAGHAVAN, B. TCP fast open. *ACM CoNEXT* (2011).

[23] RIPE NCC. RIPE Atlas. https://atlas.ripe.net/. [Online; accessed 14-March-2017].

[24] SANFILIPPO, S. hping. http://www.hping.org/. [Online; accessed 07-March-2017].

[25] SEHATI, A., AND GHADERI, M. Network assisted latency reduction for mobile web browsing. *Elsevier Computer Networks Journal* (2016).

[26] SITESPEED.IO. Welcome to the wonderful world of Web Performance. https://www.sitespeed.io/. [Online; accessed 07-March-2017].

[27] SIVAKUMAR, A., GOPALAKRISHNAN, V., LEE, S., RAO, S., SEN, S., AND SPATSCHECK, O. Cloud is not a silver bullet: A case study of cloud-based mobile browsing. *ACM HotMobile* (2014).

[28] SIVAKUMAR, A., PUZHAVAKATH NARAYANAN, S., GOPALAKRISHNAN, V., LEE, S., RAO, S., AND SEN, S. PARCEL: Proxy Assisted BRowsing in Cellular Networks for Energy and Latency Reduction. *ACM CoNEXT* (2014).

[29] THE NIELSEN COMPANY (US), LLC. Nielsen provides topline U.S. web data for March 2010. https://goo.gl/DNvXTF. [Online; accessed 14-March-2017].