Client-Centric Content Delivery Network

Sipat Triukose
International School of Engineering (ISE)
Chulalongkorn University
Bangkok, Thailand
Email: sipat.t@chula.ac.th

Michael Rabinovich
EECS Department
Case Western Reserve University
Cleveland, USA
Email: michael.rabinovich@case.edu

***Abstract*—Content delivery networks (CDNs) carry a large portion of today's web traffic. Any improvement in their performance would have a direct impact on Internet users' experience. We propose a client-centric approach to improve the content delivery performance of CDNs with minimal alteration of the current CDN platform. A preliminary evaluation of our approach based on traffic traces from a large organization network shows significant promise, with around 22%-36% performance improvement for HTTP object downloads.**

I. Introduction

Content delivery networks have become an integral part of the Internet infrastructure. Just the leading CDN provider – Akamai – claims to be delivering 15-30% of all Web traffic [8], and there are dozens other CDNs as well. CDNs are content-provider centric: they provide service to subscribing content providers and they answer to their subscribers for the performance of the outsourced content delivery. In particular, as part of this business relationship, the CDN and a subscriber agree on a subset of edge servers and locations to be used for delivering the subscriber's content. Because different subscribers may be assigned to different edge servers, and due to load balancing, users from the same organizational network, or even the same user, may be directed to different edge servers for different downloads.

This work explores a somewhat different operation model, where instead of allocating edge servers to groups of content providers, the CDN "assigns" edge servers to (groups of) organizational or metropolitan networks to which the users – the consumers of the content – belong. This brings the usage of an edge server closer to that of a client-side forward proxy, except that the edge server is still utilized only for the content from subscribers and is physically part of the CDN platform. In other words, users from a given organizational network would be directed to the same edge server to download content from *all* the CDN subscribers, while at the same time content-provider centric nature of the CDN does not change.

Our idea is inspired by an observation from our previous work that a client can download any CDN-outsourced object from any edge server, whether or not this edge server is assigned to the object's content provider, and that this edge server will cache the object for future use [13]. We further show in the present work that a server that provides good download performance to a given client is likely to continue to be a good choice for an extended period of time. With these observations, clients in a given organizational network can be "pinned" to the same edge server for all downloads of CDN-accelerated content, opening up a number of possibilities for performance improvement including (a) reduction of the overhead from frequent DNS queries to remote DNS servers; (b) better reuse of existing TCP connections hence a reduction of TCP connection start-up overheads; and (c) improved TCP connection utilization, with consequent improvement in TCP throughput due to more effective bandwidth probing and less chance of a congestion window reset after a long idle period. This paper quantifies, and sketches architectural approaches that could realize, these benefits.

A potential concern with this approach is that prolonged pinning of clients to the same edge server may hamper CDN's ability to quickly react to edge server failures or sudden overloads and other changes. As discussed in the paper, this concern is effectively addressed by periodic polling of the edge server to check its responsiveness; by selecting the polling period to be equal to the time-to-live returned by CDN's DNS responses, the agility of our approach remains the same as in a traditional CDN.

II. Related Work

Our approach is related to several past ideas that blurred the boundaries between forward proxies and CDNs, such as the Content Bridge Alliance [5] and an Akamai patent disclosing a possibility of deploying an edge server inside an enterprise intranet [9]. Unlike these efforts, we propose to improve CDN performance simply by changing the way edge servers are selected. Poese et al [11] proposed a solution where a CDN interacts with client ISPs to obtain distance information between clients and edge servers only available to the ISP. In contrast, our approach improves performance by avoiding excessive edge server churn.

Our approach tries to increase efficiency of content delivery by reducing DNS resolution, TCP start-up overheads, and increasing TCP connection reuse. One could also mask DNS resolution and some TCP start-up overheads through prefetching. Implications of DNS prefetching, and to less extent TCP connection pre-opening, have been studied [3], [12], [4] and many web browsers today offer the DNS prefetching as their standard feature. Our approach differs from prefetching in that it does not run the risk of unnecessary work.

HTTP/2 [7] has introduced a number of measures to improve performance of Web downloads, including those aimed at increasing TCP connection utilization. Our approach requires no changes to HTTP. Its evaluation in the context of HTTP/2 is a topic for future work.

III. The Approach

CDN subscribers outsource their content delivery to the CDN by replacing hostnames in their URLs with hostnames belonging to the CDN domain, typically by means of DNS redirection or URL re-writing. (We refer to these hostnames as "CDN-outsourced" hostnames.) Then, users' DNS queries for the outsourced hostnames arrive to the CDN's authoritative DNS servers, which select edge servers for each query and return their IP addresses to the users.

Our basic approach is for an organizational network to, most of the time, direct all its users to the same edge server, for all content served by a given CDN. We describe and evaluate our approach using Akamai, the leading CDN provider, as an example.

To follow our approach, an organization would modify its local DNS server (LDNS) to maintain a list of a few Akamai edge servers, which the LDNS can do by periodic DNS queries for Akamai-outsourced hostnames. When processing DNS queries from the users, the LDNS would recognize queries for Akamai-outsourced hostnames and, instead of forwarding these queries in a usual way, simply respond with the IP address of an edge server of its own choosing, regardless of the specific hostname being resolved. Normally the same edge server would be returned for all queries unless its performance degrades or it becomes unavailable. The LDNS can detect any such issues by periodically probing the chosen edge server, e.g., by requesting a small HTTP object from it. In fact, as explained in Section III-B, by choosing the probing interval to be equal to the TTL of DNS responses from the CDN, this approach retains the agility of the current CDN in reacting to unforeseen changes in the edge server operation.

A. Deployment Alternatives

There are two ways to realize this approach. One alternative is for the organizational network to simply override the CDN's edge server selection with its own mechanism, and we verified that existing CDNs allow such user-imposed edge server selection [13]. However, CDNs can fairly easily implement mechanisms to block the selection of an unintended edge server. Instead, the other alternative is for the CDN itself to *embrace* this approach and make the modified LDNS software available to organizations. Using this LDNS would be beneficial for both the the organizations and the CDN: the organization's users will experience better download performance when accessing content from the CDN's subscribers, and the CDN will deliver better performance to their subscribing content providers.

B. Edge Server Selection

While we leave detailed investigation of LDNS behavior regarding edge server selection for future work, as a starting point, we envision the following scheme. The LDNS would maintain a list of candidate edge servers. For that, LDNS can perform DNS resolution of one Akamai-outsourced hostname from each of a few Akamai's customer sites known to be mapped to a large number of edge servers. Presumably Akamai will return the best server among the servers allocated to each of these customers, and the LDNS server can now fetch a small number of bytes using an HTTP range request from each of these servers to select the best among them (there would be only few to check — one for each customer site). The candidate list can be refreshed and re-ranked periodically, and the results of the next section suggest this can be done very infrequently, with periodicity in the order of hours. The LDNS will then use this edge server for all client requests for any Akamai-accelerated content, while periodically monitoring the edge server performance. If a degradation of this server performance is detected, or re-ranking of the candidate list finds a better server, the LDNS will switch to another edge server.

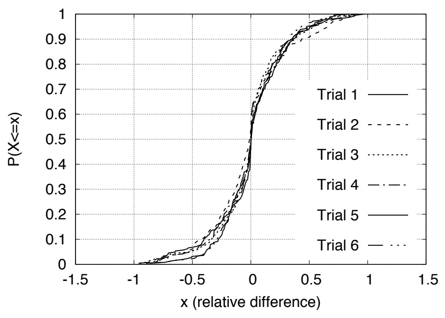


Fig. 1. Relative throughput difference. If T_r and T_a are throughput of reused and Akamai-selected servers, the relative difference is calculated as $(T_r - T_a)/max(T_r, T_a)$

Akamai's current practice bounds Akamai's responsiveness to edge server outage or overload by the TTL it assigns to its DNS responses, currently 20s; indeed, these responses will be cached and reused for the TTL period by the LDNS and browsers no matter the change in the edge server status. By using the same TTL for its responses to clients and for re-probing the selected edge server, the LDNS in our scheme will have the same agility.

IV. The Effect of Infrequent Server Selection

The potential downside of our scheme is that, because the clients reuse the same edge server for a long time, the client's download performance could degrade if network conditions change. To assess this effect, we conduct the following experiment using the DipZoom measurement platform [6]. We choose a $47,727$ byte objectfrom buy.com outsourced to Akamai. At the beginning, we request 260 DipZoom measuring points (MPs) world-wide to perform domain name resolution of "ak.buy.com" to get Akamai selected edge server IP addresses for each MP.

We then perform six trials spaced 30 minutes apart. In each trial, we request each MP to download the object (using the *curl* tool) from the edge server obtained a-priori (using server IP address directly), and from the edge server selected normally by Akamai at the time of the download (i.e., obtained by a DNS query to Akamai just prior to the download). We make sure (by pre-requesting the object) that the requesting content is cached on both edge servers prior to both downloads to avoid potential cache-misses skewing our results.

Figure 1 shows, for each trial, the cumulative distribution function (CDF) of the relative difference of download speeds (as reported by curl) between the a-priori selected server and the server selected at the time of the test, across all 260 MPs. There are six curves – one for each trial. The result shows no discernible difference between reusing the server and selecting the server at the time of measurement. The relative differences are clustered around 0, and each scenario outperforms the other in around 50% of the MPs; furthermore, this result remains consistent throughout all the trials[1], or 2.5 hours since the selection of the a-priori server. In other words, content delivery performance from the same edge server remains consistent over the extended period of 2.5 hours.

Thus, while LDNS in our approach will probe the selected edge server every 20s (which is the TTL of Akamai's DNS responses) to react to any sudden changes in the edge server status with the same agility as Akamai today, one can expect that most of the time, the clients will be able to remain pinned to the same edge server for a long time. We leave detailed investigation of LDNS behavior to future work.

V. Preliminary Evaluation

We conduct a preliminary evaluation of the impact of our approach on HTTP download performance of Akamai-delivered content. After describing our dataset, we first present a simulation experiment to assess the amount of the reduction in remote DNS queries (i.e., the queries that miss in LDNS cache and involve interaction with an authoritative DNS server) and the increase of TCP connection reuse, the main motivating factors of our approach. We then perform a replay experiment to evaluate the actual performance improvement. More details on our study can be found in [14].

A. Data Set

We use an HTTP packet trace collected at line speed from the two border routers connecting the CWRU campus network with the Internet. The trace spans 2.5 hours during the high traffic period (in the afternoon) of September 14, 2010 and comprises over 8.2 million HTTP downloads from 21,616 clients. Since we focus on performance impact to Akamai due to our approach, we extract the Akamai traffic involving Case local clients from this trace. The Akamai traffic includes 2.3 million HTTP downloads (around 28% of all HTTP requests, in the line with Akamai's claim of delivering 15–30% of HTTP traffic) from 5,725 Case clients over 0.78 million TCP connections.

Akamai deploys edge servers locally within some enterprise networks, and CWRU campus network is one of them. We refer to such edge servers as "local edge servers". While local edge servers are located within Case network, their traffic fortunately traverses the border routers due to network organization and is captured in our trace. In our Akamai trace, traffic to local edge servers represents 1.6 million requests over 0.52 million TCP connections, with the total downloaded content size of 51 GB. Off-campus edge servers are responsible for 0.7 million requests, 0.26 million TCP connections, and 35 GB of total downloaded content.

B. Simulation Experiment

1) DNS Queries: We first assess the performance improvement from virtually eliminating remote DNS queries. To this end, we need to estimate the number of LDNS cache misses in the current Akamai approach and the time taken by each query. Since our trace contains no DNS traffic, we use trace-driven simulation to recreate the stream of DNS queries issued by LDNS to the Akamai platform, by simulating LDNS cache with 20 second TTL. Initially the LDNS cache is empty. For each HTTP request from the trace, if the requested hostname has been placed into the LDNS cache within the past 20 seconds, no query is issued, otherwise the query is generated and the hostname is placed in the LDNS cache.[2] As the result, we estimate that the clients experienced 138,237 LDNS cache misses. These misses occur over 2.3M HTTP requests and 0.78M TCP connections, showing high DNS cache hit rate (over 93% for requests and 82% for connections occurred without an external DNS lookup) for Akamai-delivered content despite short TTL.

We further separately measured the response time of these misses by resolving hostnames from the trace from a machine on the Case campus network. To exclude queries that hit in the CWRU LDNS cache, we only consider queries that returned responses with TTL 20s – the authoritative TTL assigned by Akamai DNS, obtaining usable query times for 4,092 out of 5,218 Akamai-outsourced domain names in the trace[3]. We found the average cost of a miss (weighted by relative frequencies of name appearances in the trace) to be just over 22ms. Along with the low cache miss rate, the impact of these misses appears minimal (the amortized cost per connection is in low single milliseconds). In other words, the benefits of our approach due to eliminating remote DNS queries is negligible.

2) TCP Connection Reuse: We now compare TCP connection reuse in the current Akamai setup and our approach, as measured in the number of HTTP requests delivered over one connection. For this experiment, we removed all traffic to Akamai that originated from the CWRU's guest wifi gateway. Traffic from this gateway appears to come from a single client,

[1]Across all the MPs and trials, median, average, and standard deviations of the throuput ratios of a-priori over kamai-selected servers are 1, 1.17, and 0.05 respectively.

[2]In practice, web browsers and LDNSs sometimes cache the DNS query answers longer than assigned TTLs [10], although this practice seems to become less prevalent [2]. Therefore, our simulation could be viewed as an upper bound of DNS query overhead.

[3]In the hindsight, we should have repeated queries for the names resolved from the LDNS cache, but we do not believe the missed queries affect our findings.

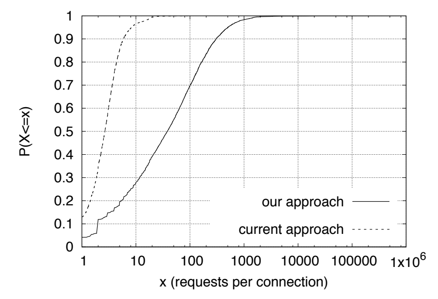


Fig. 2. Per-client connection reuse (requests/connection) in both approaches

while in fact coming from a large number of users behind the gateway, and it can skew the results. After removing this traffic, we are left with $1,615,150$ HTTP requests for Akamai-outsourced content over $495,213$ connections in the current approach.

For the current setup, we obtain connection reuse directly from the trace. The reuse in our approach is simulated as follows. In a separate study, we measured persistent connection keep-alive in Akamai egde servers to be 500 seconds [15] (which is notably longer than typical values on websites [1]). Then, we consider a set of HTTP requests from the same client with inter-request gap not exceeding 500 seconds, regardless of which connection they belong to in the trace, to utilize the same TCP connection in our simulation. Any gap of 500s or more opens a new TCP connection. As a result, our approach only requires $9,435$ to carry the 1.6 millions requests.

Figure 2 shows the CDF of the average connection reuse per client in the current and our approach, measured in the average number of requests per connection, for all connections from a given client. It shows a dramatic increase in connection reuse in our approach. Overall, one TCP connection in our approach on average transfers 171.19 HTTP requests compared to only 3.26 or 3.82 requests in the current approach (depending on whether or not we exclude connections that transfer no data from the $495,213$ connections in the current approach). Next section assesses how this improvement translates into the end-to-end download performance.

C. Replay Experiment

We compare the end-to-end performance of the current and our approaches by replaying actual Akamai traffic from the trace. In particular, this quantifies the impact of higher connection utilization on throughput of content downloads.

Since it is infeasible to replay the entire traffic from our trace, we randomly select 30 clients accessing Akamai-accelerated sites and extract the Akamai traffic trace generated by these clients to use for our replay experiments. This subset of the Akamai trace represents 4,717 HTTP requests over 1,852 TCP Connections and constitutes 549 MB of content; There are 3,186 requests to local Akamai edge servers over 1,251 TCP connections and 1,531 requests to off-campus Akamai edge servers over 601 TCP connections; The internal and external requests constitutes 533 MB and 16 MB of Akamai content respectively.

1) The Replay Method: We perform two replays of the clients' requests, using our approach (*our-replay*) and under the current Akamai practice (*current-replay*). For simplicity, and because most browsers turn pipelining off for security reasons, neither replay use HTTP pipelining. Thus, two overlapping downloads will be serialized as two successive back-to-back requests in the order of their starting timestamps in the trace. At the same time, consecutive non-overlapping downloads in the trace are replayed while preserving the original inter-request intervals in the trace, except if a request during the replay does not finish before the designated time of the next request, in which case we start the next request right after the completion of the first one. During a replay, we record the domain name resolution time, every connection establishment time, and content transfer time.

In *our-replay*, we open a TCP connection to our designated edge server and attempt to utilize this connection for requests across all customer contents. If the connection is terminated by the remote host, we re-open another connection to the same edge server.

In *current-replay*, for each client, we group requests by the TCP connection to which they belong. For each group of requests, a client opens a TCP connection to the designated edge server and sends all these requests through this connection, in a similar way to *our-replay* (no pipelining, otherwise preserving inter-request time gaps). Once this group of request is completed, the connection is closed and the new connection, for the next group of requests, is opened in immediate succession to the same edge server.

The above technique serializes HTTP requests while largely retaining transfer timing within each TCP connection.

We account for DNS query overhead as follows. In *our-replay*, for each client, we perform the domain name resolution for the first Akamai-accelerated URL in the trace to record the time of getting the designated edge server. However, *current-replay*, distorts timing of the initiation of the TCP connections and thus makes recreating and replaying DNS queries more difficult. Therefore, in *current-replay*, instead of performing the DNS query before every TCP connection, we replay the DNS queries separately according to the original timing of the starts of the TCP connections to ensure that the gaps between all DNS queries are the same as in the real trace. Since we replay these queries through production LDNS, their timing reflects natural caching effects in LDNS at the time of the experiment.

We evaluate our approach in two scenarios, in an organization that has local Akamai servers deployed and an organization that does not. In the first first scenario, we use a local Akamai edge server on Case campus as the selected edge server in the replays, for both *our-replay* and *current-replay*. In the second scenario, we choose the most popular external Akamai edge server used by the Case community as our selected edge server. We refer to two replays in each

	Current	Our	% $\frac{Our}{Current}$
DNS time	24.81	0.41	1.67
TCP Handshake time	827.98	42.06	5.08
Transfer time	3267.67	3179.17	97.29
Total Download time	4120.47	3221.64	78.19

TABLE I
REPLAY TIMES (IN SECOND) OF *int-our-replay* AND *int-current-replay*.

	Current	Our	% $\frac{Our}{Current}$
DNS time	18.08	0.24	1.33
TCP Handshake time	1192.31	63.38	5.32
Transfer time	3885.31	3200.21	82.37
Total Download time	5095.70	3263.83	64.05

TABLE II
REPLAY TIMES (IN SECOND) OF *ext-our-replay* AND *ext-current-replay*.

scenario as, respectively, *int-our-replay*, *int-current-replay*, *ext-our-replay* and *ext-current-replay*.

We used a host on Case campus in all HTTP replays as well as for DNS replays in *int-current-replay*. For *ext-current-replay*, we cannot replay the DNS queries from the Case client because there exists a local Akamai authoritative DNS server on the Case network. Therefore, we perform this DNS replay in another network, where Akamai has no presence, using the PlanetLab node at the University of Chicago for this purpose.

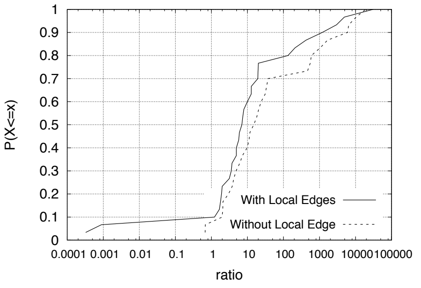
2) Results: Tables I and II summarize the comparison between total download time of the same contents in the current network configuration and in our proposed configuration. The tables also break down the total time into DNS query time, TCP handshake time, and transfer time, showing the contribution of each component to performance improvement.

Overall, as seen from Tables I and II, clients take around 69 and 84 minutes to download all the contents in the current configuration as opposed to around 53 and 54 minutes with our approach, for the network with and without local Akamai edge server deployed respectively. With 4,717 HTTP requests in the trace, this translates into improvement of our approach in average object download times from 0.87s to 0.68s in the network with a local edge server and from 1.08s to 0.69s without, 21.81% and 35.95% improvement.

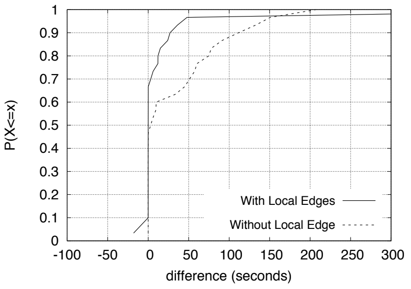
The root cause for these gains is improved TCP connection reuse in our approach. Table III compares the number of connections and connection reuse in our and current approach. Note that the number of connections in the current approach slightly exceeds that in the trace because a few groups of HTTP requests that were transferred over a single connection in the trace required several connections during the replay. For the clients studied, our approach achieves more than 20 and 24 times higher connection reuse than the current approach

With local edge server	Current	Our
# of Connections	1856	91
Connection Reuse (average request/connection)	2.5	51.8
Without local edge server	**Current**	**Our**
# of Connections	1857	77
Connection Reuse (average request/connection)	2.5	61.3

TABLE III
TCP CONNECTION REUSE IN THE NETWORK WITH AND WITHOUT A LOCAL AKAMAI EDGE SERVER DEPLOYED.



(a) Cur-to-our ratio (greater than 1 means our approach is better)



(b) Cur–our difference (positive means our approach is better)

Fig. 3. The comparison between per-client total TCP handshake time spent in the current and our approach.

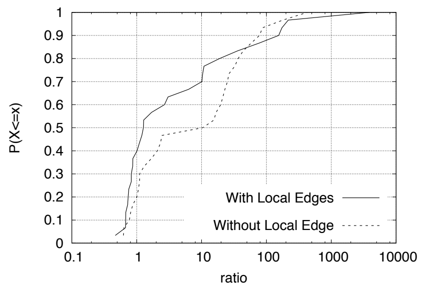
in the network with and without local Akamai edge server respectively.

Turning to the components comprising the full object download time, Tables I and II show that while our approach virtually eliminates the overhead of remote DNS queries, their impact on the overall performance is negligible. This confirms the simulation results of Section V-B1. The contribution of the TCP handshake savings and of the reduced transfer times are more significant.

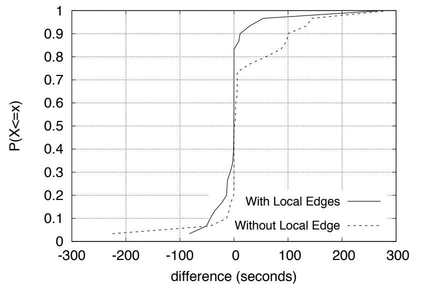
To show these impacts for individual clients, Figure 3 presents the CDFs of the ratios and absolute differences between per-client total TCP handshake spent respectively in the current approach and our approach. Figure 4 shows the same graphs for the per-client total transfer times.

Figure 3a shows that our approach reduces TCP handshake overhead for more than 90% of the clients, both with and without local Akamai servers. In fact, 40% and 60% of the clients in networks with and without local Akamai server respectively spend at least 10 times more on handshake in the current approach.

As for transfer times, Figure 4a demonstrates that most clients (around 60% in the network with, and 80% without, a local edge server) reduce their transfer time with our approach. In fact, for many clients, the benefits are very significant: 50% of clients without a local edge server, and 30% of clients with a local edge server experience at least an order of magnitude improvement. Why then the overall transfer time improvement shown in Tables I and II is relatively modest? Figure 4b provides the answer. It shows that the absolute difference in the time they spend transferring objects is generally modest.



(a) Cur-to-our ratio (greater than 1 means our approach is better)



(b) Cur–our difference (positive means our approach is better)

Fig. 4. The comparison between per-client total transfer time spent in the current and our approach.

In other words, those clients that experience dramatic relative performance improvement generally download small objects and therefore, their absolute transfer times are generally small and they contribute less to the total transfer time of all clients. Meanwhile, clients that download large objects do not benefit from improved connection reuse as much because large downloads by themselves are able to utilize TCP connections well and amortize the connection startup costs. Yet these large downloads contribute a disproportional share to the total transfer time.

To recap, in our replay experiment, our approach shows around 36% improvement in average object download time for a network where no Akamai edge server is deployed locally. Even in a network where there is a local edge server deployed, our approach still brings around 22% improvement in object download times.

VI. Discussion and Future Work

In this study, we identify optimization opportunities in the current practice of co-location CDNs and propose a solution, which requires very small changes to the client network, to leverage these opportunities. Our experiments quantify performance gains Akamai would have obtained if this approach was deployed on the CWRU campus network. A limitation of our study is that it is unable to account for possibly higher hit rate at edge servers in Akamai's current approach, due to a limited set of servers used for a given website. We do not believe this has a significant effect on our findings. According to our study, from the client perspective, Akamai will achieve significant performance benefits with our approach.

However, it raises important questions for further study. Although our approach requires no change in the Akamai platform, it affects how Akamai's resources are utilizes. How should Akamai adjust its internal system management if it is to embrace our approach? A client site can unilaterally adopt our approach today and gain benefits for its users. But how would the performance of the platform as a whole be affected if many clients adopt our approach? Finally, a proof-of-concept implementation of our approach and live evaluation with actual users would be the next step towards a wide adoption of our approach. We leave this effort for future work.

VII. Conclusion

We propose a simple approach to improve the performance of downloads of CDN-accelerated content from the user perspective, which client networks can adopt with or without cooperation from the CDNs. By directing a client to the same edge server for all content accelerated by this CDN, our approach dramatically increases TCP connection reuse between the client and the CDN platform. While many questions remain for future investigation, our preliminary study shows a significant promise of our approach as a "low hanging fruit" to achieve a tangible performance improvement in the download performance of CDN-accelerated content.

Acknowledgent: We are indebted to the CWRU ITS organization, especially Jim Nauer, Kevin Chen, Dan Matthews, and Roger Bielefeld, for their help in procuring data for this study.

References

[1] Z. Al-Qudah, M. Rabinovich, and M. Allman. Web timeouts and their implications. In *Conf. on Passive and Active Measurement*, pages 211–221, 2010.

[2] T. Callahan, M. Allman, and M. Rabinovich. On modern DNS behavior and properties. *ACM SIGCOMM Comp. Comm. Rev.*, 43(3):7–15, 2013.

[3] The Chromium projects – DNS Prefetching. http://dev.chromium.org/developers/design-documents/dns-prefetching.

[4] E. Cohen and H. Kaplan. Prefetching the means for document transfer: a new approach for reducing Web latency. *Computer Networks*, 39(4):437–455, 2002.

[5] Inktomi, Adero and America Online announce strategic content and technology distribution agreement. http://www.timewarner.com/corp/newsroom/pr/0,20812,666754,00.html.

[6] DipZoom - Deep Internet Performance Zoom. http://dipzoom.case.edu.

[7] M. Thomson M. Belshe, R. Peon. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. IETF Request for Comments 7540, May 2015.

[8] B. Maggs and R. Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66, 2015.

[9] Charles J. Neerdaels. Extending an Internet content delivery network into an enterprise. Akamai Technologies U.S. Patent #7,096,266.

[10] Jeffrey Pang, Aditya Akella, Anees Shaikh, Balachander Krishnamurthy, and Srinivasan Seshan. On the responsiveness of dns-based network control. In *IMC*, pages 21–26, 2004.

[11] Ingmar Poese, Benjamin Frank, Bernhard Ager, Georgios Smaragdakis, and Anja Feldmann. Improving content delivery using provider-aided distance information. In *IMC*, pages 22–34, 2010.

[12] H. Shang and C. Wills. Piggybacking related domain names to improve DNS performance. *Computer Networks*, 50(11):1733–1748, 2006.

[13] S. Triukose, Z. Al-Qudah, and M. Rabinovich. Content delivery networks: Protection or threat? In *ESORICS*, pages 371–389, 2009.

[14] Sipat Triukose. *A Peer-to-Peer Internet Measurement Platform and Its Applications in Content Delivery Networks*. PhD thesis, Case Western Reserve University, 2014.

[15] Sipat Triukose, Zhihua Wen, and Michael Rabinovich. Measuring a commercial content delivery network. In *The 20th Int. Conf. on World Wide Web*, pages 467–476, 2011.