Rapid IP rerouting with SDN and NFV

Jeffrey Lai, Qiang Fu
School of Engineering and Computer Science
Victoria University of Wellington
Kelburn, Wellington
New Zealand
Email: {Jeff.Lai, Qiang.Fu}@ecs.vuw.ac.nz

Tim Moors
School of Electrical Engineering and Telecommunications
University of New South Wales
Sydney, NSW 2052
Australia
Email: t.moors@unsw.edu.au

***Abstract*—Current Content Delivery Networks (CDNs) primarily utilize the Domain Name System (DNS) to direct users towards optimal content replica servers. Typically, these CDNs will leverage specialized DNS servers to perform load balancing, by providing different users with different IP endpoints for a particular hostname. However, there typically exists some form of DNS cache between the user and the authoritative DNS for a particular web service (Examples include web browsers, operating systems, and home routers). While these DNS caches typically improve user experience and network performance by reducing the number of redundant DNS requests, there exist scenarios where the presence of a DNS cache can actually *harm* network performance and user experience. For example: if a server or datacentre were to suddenly become unavailable, end users may become 'stuck' trying to access the old IP address contained in their DNS caches, even though the authoritative DNS server may have updated it's records accordingly. In order to address this issue, we propose a system that enables network operators to transparently redirect users towards optimal servers during times of network congestion or high server load. By leveraging SDN and NFV, we are able to implement this system over a variety of network infrastructures, and to various levels of scale. Our initial tests indicate that our methodology has the capability to vastly improve user experience by several orders of magnitude during times of high network load, over a variety of network conditions. Our system is application-layer agnostic, and is compatible with various networking protocols.**

***Keywords*: CDN, SDN, NFV, NAT, TCP, QOS, OpenFlow, Traffic Engineering, Click software router**

I. Introduction

Content delivery networks are finding an increase in usage in today's internet landscape. In response to a growing demand for high definition and high bandwidth content, internet service providers have started to host their own CDNs[1]. Large CDN providers such as Google and Akamai now also collaborate with ISPs in order to provide better services both for the end consumers of content, and the content producers themselves. Typically, these ISP operated CDNs[1] are in a unique position wherein they control the edge routers that are topologically closest to their users. Because of this, ISP operated CDNs have direct access to a larger amount of network information that may help in making traffic engineering decisions. ISP operated CDNs are also capable of performing traffic engineering operations at any point on their network, as they control the network infrastructure that user traffic is passing through. This allows them to perform traffic engineering to a greater degree than traditional CDNs that do not have this level of control on user traffic.

Currently, hostnames for CDN hosted websites and content are resolved using DNS techniques that provide users with an IP address that maps to an optimal CDN node for that user at the time of request[2]. While current techniques work well for providing users with optimal IP addresses at time of request, future changes to network and/or server conditions do not get communicated to the end user until they next perform a DNS request. Current authoritative CDN DNS servers attempt to get around this by setting low time to live values in their DNS responses. However, most major browsers tend to ignore these TTL values, and will cache incoming DNS information for a minimum of one minute[3]. Other DNS resolvers in the network may also perform DNS caching, leading to stale DNS information being used for much longer than intended. This has the effect of delaying the time it takes for updated DNS information becoming useful for end users, regardless of how fast a network operator can detect and resolve such network issues.

Consequently, we have designed a solution that leverages software defined networking(SDN) and network function virtualization(NFV) concepts to rapidly and transparently redirect users from nonoptimal to optimal endpoints. Our solution utilizes OpenFlow[4] along with the Ryu OpenFlow controller[5] in order to flexibly and dynamically redirect traffic to our NFV IP-rewriter. Our NFV IP-rewriter has been written using the Click modular software router[6]. Results obtained in our test network indicate that even with minimal DNS caches present in the system, our solution is capable of vastly improving content access delay times. With modern browsers caching DNS information for at least one minute, removing this 'caching penalty' has the potential to vastly improve user browsing experience in times of network congestion. By employing our proposed IP rerouting architecture, SDN-enabled ISP-CDNs can thus make use of more responsive network decisions, such as rerouting between CDN datacenters as dictated by network and/or load requirements. Our solution is successful in transparently performing IP redirections on end-user traffic, and is capable of respecting pre-existing TCP

[1]In this document, the term 'ISP operated CDN' refers to both CDNs wholly owned and run by an ISP, as well as ISP CDN collaborations.

978-1-4799-5952-5/15/$31.00 ©2015 IEEE

traffic flows. In this paper we present and evaluate our working proof-of-concept solution.

Our paper is organized as follows. In Section II, we examine some related work in the areas of using SDN and NFV concepts for CDN optimization. In section III, we discuss the requirements of our proposed SDN/NFV IP-rerouting scheme. Subsequently, we present our IP-rerouting architecture in Section IV. In Section V, we discuss our results and findings. Finally, Section VI concludes our paper.

II. Related work

Much work has been done on improving CDNs in the last decade[7], [8], due to the large effects they have on increasing the efficiency of content delivery. However, despite the fact that DNS caching has been employed since the start of the rising popularity of CDNs, published research on the effects of DNS caching by end users has not yet been explored in depth. Work done by Akamai[1] on CDN-ISP collaboration has shown that a DNS based system can adequately perform user assignment in ISP CDNs, and suggests that researching improvements to the DNS IP-redirection system is a worthy pursuit.

While we are primarily concerned with DNS-based IP-redirection methods for CDNs in this paper, another mechanism that exists and is widely used to perform CDN request redirection is utilizing IP anycast[9]. Current IP anycast implementations can still suffer from route-flapping and requires complex BGP maintenance to be able to react to network changes[10]. Research by Yulei Wu et al.[11] demonstrates an SDN implementation of IP anycast that overcomes this issue, however suffers from the need for a large flow table size in order to not break TCP connections. This requirement for a large table size is explained in Section III.

OpenCache[12], an SDN caching platform has recently been developed and is currently under research. While they employ similar redirection techniques to the ones detailed in this paper, theirs is purely an SDN solution, and is more focused on providing a caching system for high quality streaming video.

Research by Justine Sherry et al.[13] indicates that it is indeed practical and feasible for enterprise networks to outsource middlebox processing to the cloud. They also discuss the viability of utilizing DNS-based redirection to provide access with multiple cloud PoPs. Scenario C in Section V is a test scenario we created that simulates 'spinning up' a CDN server in the cloud in response to high CDN server load. Their research also suggests that implementing the NFV IP-rewriter portion of our architecture in a cloud network is a feasible possibility.

The NFV portion of this work heavily utilizes the Click software router. Our network tests were performed with the Click software router and it's provided modules, as well as some customizations that we developed ourselves to allow for dynamic NAT remapping. Recent research by Joao Martins et al.[14] provides optimizations to the Click routing software, and provides a custom operating system with a virtualization shell that allows for rapid NFV VM initialization. They have achieved good real-world hardware results with their research so far, capable of performing middlebox functions at high bandwidth (10Gbps). Our future research will involve utilizing the optimizations provided by their research.

III. Requirements for an IP-rerouting scheme

As stated in Section I, the prevalence of DNS caching leads to slower uptake of updated DNS information across the network. Current browser usage statistics provided by w3schools[15] indicates that the two most popular web browsers (Google Chrome and Mozilla Firefox) currently hold just over 85% market share. These browsers currently cache DNS information for at least one minute, regardless of TTL[3]. If we take these as the only DNS caches present in an end user's system, then on average it will take 30 seconds for changes to DNS information to propagate down to the end user. Further caches between the user and their CDN provider will only exacerbate this effect. Thus, it is required that the ISP-CDN IP-rerouting scheme be able to overcome this DNS caching issue.

For our proposed IP-rerouting scheme to be successful, it should also require zero user-side configuration, much like current DNS CDN redirection schemes. Because the majority of traffic from a CDN is delivered via TCP/UDP flows, our system must also be able to preserve sessions where possible. That is, we should avoid redirecting a TCP flow mid-transfer if possible, instead preferring to redirect entire TCP flows from start to finish. This creates a requirement that our system also needs to be able to handle large amounts of state information, so as to avoid inadvertently breaking TCP session state.

IV. SDN and NFV based IP-redirection

Software defined networking via OpenFlow provides network operators with the ability to flexibly and rapidly apply changes to how switches forward traffic[4] However, because we need to perform IP-redirection whilst maintaining TCP session state (As mentioned in Section III), this can lead to unreasonable flow table sizes and switch memory required in the OpenFlow-supported switch, which in turn inflates the cost required to build and provision such a network[16]. Therefore, it is desirable to implement the stateful part of our solution in a more scalable manner, that is agnostic towards the amount of state that is required to be held. To address this, we implement the IP-redirection part of our scheme via a separate, virtualised server running some form of software router. For this purpose we utilized the open source Click software router[6]. The Click software router is modular and can easily be used in a virtualization setting, and so is suitable for our NFV IP-redirection requirements. Our proposed architecture is thus composed of three parts:

A. Global knowledge and control

In this context, 'Global knowledge and control' means knowledge and control of the entire ISP-CDN network. Our system assumes that the network operator has access to, and control of the entire network. Additionally, we also assume

that the network operator similarly has a global view of the current network status. Access to this information is required in our proposed architecture because otherwise, the operator has no information to act upon, thus rendering our entire problem moot. In general, a network utilizing SDN principles will naturally have good access to global knowledge and control, as current SDN paradigms have been designed with those goals in mind[4]. We are currently also developing an API that provides easy access to the global network information provided by OpenFlow networks to aid our research.

This network information is used by our system to decide when/where to perform IP-redirection towards. For example, if the CDN Server typically serving geographic location *A* has crashed, then we wish to redirect users in area *A* away from that server. The ability to make this kind of meaningful decision requires, by definition, global knowledge and control of the operator's network. Because our proposed architecture only requires the capability of having global knowledge and control, we have not specified a specific method in which to obtain said knowledge and control. In our network testing, we made IP-redirection routing decisions based on having such assumed knowledge.

B. OpenFlow control

We use OpenFlow to control the movement and logical control of our network traffic. Specifically, in our architecture, we use it to identify traffic going from a desired set of users[2] towards certain specific CDN replica servers. Traffic identified as fitting these criteria is then sent towards a prespecified IP-redirection device. If the entire network is OpenFlow enabled, it is possible for us to apply these flow control methods anywhere in the network. This allows the network operator to perform traffic engineering in the most optimal manner. In order to perform OpenFlow control, we use the Ryu OpenFlow controller in our solution, but in theory any controller/scheme that provides suitable global network access, knowledge, and control would also work. Table I below gives an example of what the typical OpenFlow rules required in our system look like. Of note is the fact that we can match on both TCP port 80 and 443, allowing us to perform different IP-redirection strategies on encrypted vs unencrypted CDN TCP flows. It is important to note that while OpenFlow is capable of rewriting the source and destination IP addresses of packets by using the 'set field' action, we do not utilize OpenFlow in this manner. This is due to the fact that an OpenFlow switch would then have to create an individual flow rule for every TCP flow thus processed in this manner, which could quickly lead to a flow table explosion[3]

[2]For example, all users from a certain network location, who may all share the same subnet

[3]In regular usage, a single web browser can easily open more than 10 TCP sessions to a server, each of which would require an individual flow in our proposed architecture

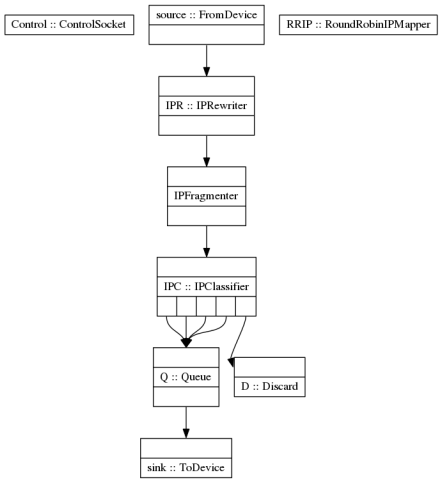


Fig. 1: Example Click configuration

C. NFV IP-redirection

Our IP-redirection requirement is achieved via Network Function Visualization. Instead of using a dedicated middlebox to perform the IP-redirection, we instead virtualize the network function we require. This allows us to then run this virtualised routing software at whatever scale we require - be it a virtual machine running in the cloud, or a dedicated server made from off the shelf hardware. For example, in response to an increasing bandwidth or computation requirement, we can scale out by increasing the number of NFV VM instances. We have used the Click open source software router to implement this NFV part of our architecture. Software routers built with Click are typically constructed from multiple 'modules', which each perform one action, such as classifying a packet based on it's IP address, or rewriting a packet's destination IP. Our NFV IP-redirection software router utilizes several of the pre-provided modules that come with Click, as well as some customizations[4]. An example of one of the modular configurations used in our testing is shown above, in Figure 1. The two separated modules, ControlSocket and RoundRobinIPMapper, allow us to dynamically configure the Click software router to perform IP-rewriting towards whatever set of IP endpoints we desire, on demand. The ControlSocket module can be tied to an API so as to allow external developers (Such as one from a CDN provider) to provide the NFV module with the requisite CDN IP endpoints. RoundRobinIPMapper also allows us to rewrite to a pool of IP addresses, if it is desired to perform a form of round robin IP load balancing. In our architecture, state information regarding client-server TCP 4-tuples are stored within this VNF, in a format similar to current NAT and load-balancing implementations. This reduces the impact of our system on pre-existing SDN-OpenFlow architectures, and allows for incremental implementation of our design. In our preliminary testing we assumed a VNF placement that was

[4]Specifically, changes to the RoundRobinIPMapper module that allow for run-time editing of NAT rules

TABLE I: Example flows installed in OpenFlow switch

Flow that catches traffic from h1 (10.0.0.1) to the CDN replicas

Priority	In-port	Protocol	IP SRC	IP DST	SRC port	DST port	Actions
10	*	TCP	10.0.0.1	10.0.1.0/24	*	80	Send packet to NAT VM

Flow that catches traffic back from the CDN replicas

Priority	In-port	Protocol	IP SRC	IP DST	SRC port	DST port	Actions
10	*	TCP	10.0.1.0/24	10.0.0.1	80	*	Send packet to NAT VM

physically close to the OpenFlow switch, such as in a nearby rack. Future testing will look at the performance impact of having the VNF placed in further locations[5]

The three above components combine together to create our overall system. Since our system only requires a minimal set of flow rules (as seen in Table I) to be installed in the OpenFlow-enabled switches that sit between end-users and CDN servers, our OpenFlow component is capable of being implemented in pre-existing OpenFlow-enabled networks. Similarly, since our IP-redirection component exists as a separate software process that can be run in a virtual machine, we can also implement that component in a pre-existing OpenFlow-enabled network. By implementing a scheme upon where the IP-space is split and directed towards corresponding separate NFV rewriters, it is easy to scale-out our NFV IP-rewriter solution.

As such, our general system process is as follows:

1) When a network/server fault is detected, identify which set of users needs to be redirected to which set of CDN replica servers
2) Start up the NFV IP-redirection software/VM/server, and provide it with the relevant IP addresses that point to the desired CDN replicas
3) Install flow rules into OpenFlow switches which match on the relevant user IP-space addresses and CDN server IP-space addresses
4) These flow rules then match on desired traffic going from the identified end users to the identified CDN servers, and vice versa.
5) Traffic that matches these rules gets forwarded to our NFV IP-rewriter
6) Our NFV IP-rewriter keeps track of TCP state, and also rewrites source/destination IP addresses accordingly
7) Packets are then sent back to the switch from the NFV IP-rewriter

The information being provided to our system components can come from a variety of sources, and may change depending on the amount of shared information between the ISP and CDN bodies that are in collaboration. To ease this information transfer (And also to allow for each party to withhold sensitive information), our system only requires information in the form of endpoint IP addresses/subnets. Further information *may* be provided by each party to enhance the advantages of collaboration, but is not necessary. In our testing, we assume a minimal amount of information has been provided by both sides, in the form of CDN server IP endpoints, and ISP client IP endpoints. This information can be provided to the switch and IP-rewriter VNF by a number of mechanisms, and has not been firmly set in our architecture.

This system process ends up identifying user traffic that benefits from being redirected towards a different CDN endpoint[6], forwarding that traffic to a virtualised IP-rewriter, and then getting the source/destination IP addresses altered accordingly, and sent back to the switch from the NFV IP-rewriter. Packets from the end-user going towards the CDN server get their destination IP addresses changed in a destination-NAT operation. Correspondingly, packets coming back from the CDN server get their source IP destinations changed in a source-NAT operation. Because we are just altering IP addresses, this process should be compatible with the pre-existing forwarding and/or routing scheme that an ISP CDN already has established. From the user's perspective, they are still communicating with the server IP address stored in their DNS cache. But from the network perspective, they are now communicating with the updated server IP address.

V. Performance evaluation

In order to test our proposed system architecture, we created a test network environment in which we set up various networking scenarios that could stand to benefit from our proposed system. Our test network was created using the mininet networking virtualization tool[17] although as previously stated, any OpenFlow/SDN network should be capable of performing these tests. The controller we have shown in our simplified network diagram in Figure 2 is a combination of a number of control elements that define the operation of both the NFV IP rewriter and our OpenFlow switch.

We first demonstrate the claim we made in Section III on the effects of DNS caching on browser performance, by performing some basic benchmark testing to see their effects first hand. Our test network consisted of a DNS server, two CDN replica servers, a basic router, and some end hosts. The DNS server was configured such that all DNS responses had a low TTL of 20 seconds, to highlight DNS caching

[5]For example, sending packets to a remote datacentre to be rewritten

[6]As defined by the network operator that has access to global network information

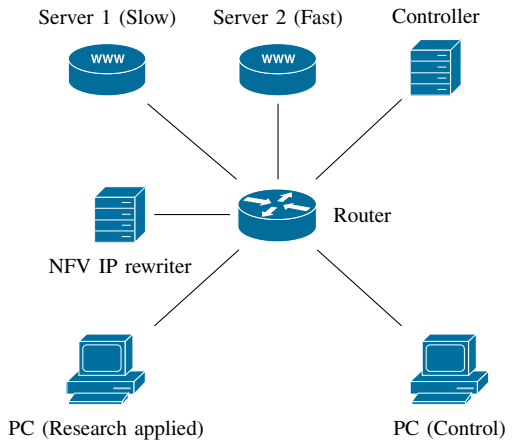


Fig. 2: Simplified test network diagram

being employed by user browsers. In our tests, end users would connect to CDN Server 1, and load content from it's hosted web page. The content provided on these webpages consisted mainly of images of varying file sizes, ranging from a few hundred kilobits in size to a few megabits. We then terminate the link between the CDN server and the end user, while simultaneously updating the DNS records in the DNS server such that the CDN hostname would point to the new server. The time taken for the end-user to access content from the newly available CDN server (CDN Server 2) was then timed. The results can be seen in Figure 3 below, in the Link failure graph under 'Without augment'. The 'Reference load time' measurement lists the time taken to load the page normally. As predicted, there is a noticeable delay of over 60 seconds until the end-user can access the new CDN server, despite updated DNS information being available. What is interesting however, is the that the two browsers we tested demonstrated different failure modes when utilizing their DNS cache. During this testing, we also simulated 'frustrated user behavior', whereupon a user would attempt to refresh the page after a certain period of inactivity. Different user behavior would result in different load times, even if we kept all other parameters of our test network untouched. With no other DNS caches present in the system apart from the browser DNS caches, these results thus act as a benchmark that our proposed architecture must meet.

Having validated our claims and problem statement, we thus present and evaluate the performance of our proposed system architecture. The three network scenarios that we tested for are as follows:

A. Network link failure

Network link failure involved having the link between a CDN server and it's neighboring router being turned off. In this scenario, it is desirable to completely redirect end-users away from the CDN server behind the failed network link, and towards a CDN server which is still available. While this would break any pre-existing TCP/HTTP sessions, this is unavoidable, and would happen anyway if the IP-redirection had not occurred in the first place. In this scenario, because we are completely redirecting end-users away from the affected CDN server, situations of both link oscillation and complete link failure are effectively handled.

B. Network link congestion

Network link congestion involved having the link between a CDN server and it's neighboring router becoming bandwidth-limited. In this scenario, similar to the one above, it is desirable to redirect users away from the CDN server behind the congested link, and towards a CDN server which is more available. In this environment, it is possible to avoid breaking pre-existing TCP connections by passing packets to the NFV IP-rewriter for a minute or two without actively rewriting IP addresses. This way, we pre-populate the NFV IP-rewriter with TCP flow information beforehand, and thus only redirect *new* TCP flows. We simulated bandwidth-limited congestion by configuring mininet parameters during testing.

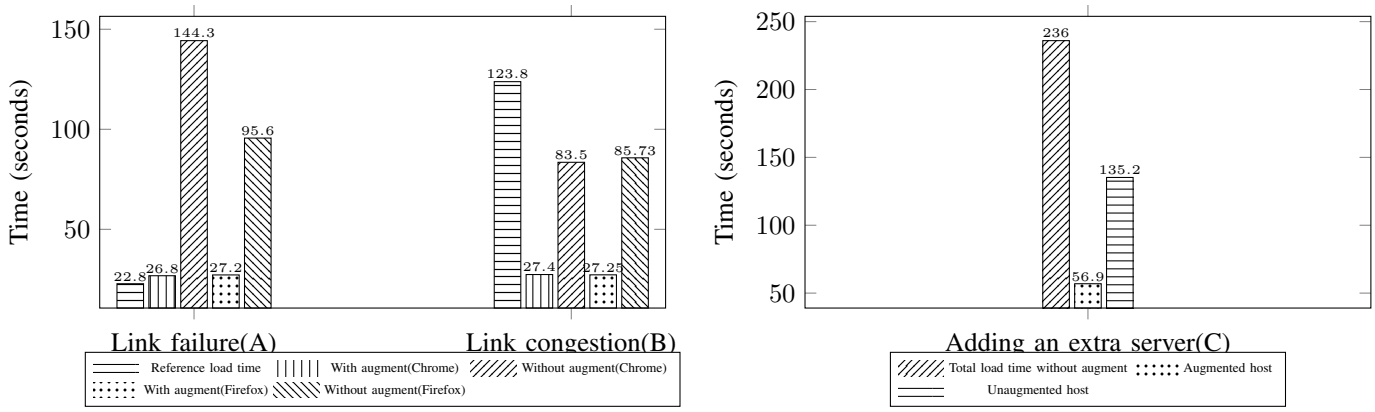
C. Adding CDN server resources

In this scenario, we start with only one CDN replica server, which is under heavy congestion, similar to test scenario B, above. Heavy congestion is simulated by limiting the outgoing link bandwidth from the CDN server down to 1Mbps. This scenario differs from scenario B by having both users (one with the augment and one without) loading the CDN resources through this link simultaneously. We then add another CDN replica server[7], and observe how long it takes for the end users to start using this updated information. Again, by passing packets to the NFV IP-rewriter for a short period of time before we actively start rewriting IP-addresses, we can avoid breaking pre-existing TCP flows. In this scenario, we are interested in observing if using our proposed system architecture allows for a rapid instantiation of extra CDN replica servers, improving performance and QoS for end users regardless of if they have our augmentation applied or not. For example, by offloading users from a CDN surrogate outside the ISP network onto a CDN-surrogate within the ISP network. This feature may be of particular interest to ISP-CDNS providing content that may become subject to flash-flood crowds, such as new-release video content.

Our test network was similar in each scenario, and can be seen in Figure 2 above. In each scenario, we tested using unencrypted HTTP traffic on TCP port 80. Users would load a website on CDN Server 1, and reach a landing page via an uncongested 10Mbps link. While connected to the landing page, a 10 second pause was induced, during which one of the three network scenarios described above were then implemented. DNS information was altered accordingly. After the 10 second pause, the end users would then click on links that led to sequential pages containing web content, which totaled 100 megabits. We then timed how long it would take

[7]While simultaneously updating the DNS records to point to this new server

Fig. 3: Test results for the three test scenarios



for the end users to reach the end of the sequential content pages. In scenarios A and B, this was done in separate tests for users which had our system augmentation applied, and those without the augmentation, so as to compare their timings separately. In scenario C, the timings displayed are for both users loading the resources simultaneously.

The results can be seen in Figure 3 above. In the first graph, for Scenario A, 'Reference load time' was the time it took to completely load the CDN resources via the 10Mbps link, without any further network changes. This measurement acts as the benchmark loading time for our ISP-CDN. Similarly for Scenario B, the reference load time is the time taken to load all the resources through the bandwidth limited 1Mbps link. We use these measurements to compare the effects of our proposed architecture. The other measurements in the first graph are the times taken to completely load the resources after applying the scenario condition, with one measurement for control(no augment) and one for the system with our architecture applied(with augment). For these two scenarios, we provide measurements for two browsers; Mozilla Firefox and Google Chrome. Due to the differences in how these two browsers handle DNS caching and HTTP/TCP connections, this allows us to observe how network performance changes with these differences in handling, despite identical network conditions.

Similarly, the 'total time without augment' measurement in the second graph(Scenario C, Adding an extra server) was for two hosts to simultaneously load the CDN resources under the congested 1Mbps link. The other two measurements were the times taken for the host with augment and the host without augment to completely load the CDN resources as described in Scenario C, after we introduced the new replica CDN server.

In all scenarios, we saw increased network performance and decreased content access delay and load times for clients that had the network augment applied. While the differences between the control and the test cases are dependent on other parameters[8], it still indicates that our proposed architecture can provide real benefits to ISP-CDNs. These tests were also performed in a system with minimal DNS caching, wherein the only DNS cache in use was that of the browser. If operating system caches (such as Linux's dnsmasq) were also deployed in our tests, it is possible that larger differences in content access delay times would be observed, due to the presence of extra DNS caches. As such, the performance increases demonstrated by our proposed architecture actually indicate the *worst case* performance scenarios for our proposed system, where minimal DNS caching is in use. Additional DNS caching would increase the difference between our system's results, and that of the control.

Performance in failure scenarios was not identical between the two browsers tested. As can be seen in scenario A: link failure, there is quite a discrepancy between the performance of Firefox and Chrome. This reinforces the notion that even though network status and information has updated, browser behavior ultimately has a large effect on how a user perceives network performance. In this case, browser behavior has influenced the usage of DNS information.

VI. Conclusion

We have presented our novel SDN and NFV based IP-rewriter that enhances pre-existing DNS CDN redirection mechanisms. Our proof of concept solution has successfully worked in our test networks by decreasing content access delay time by over a minute, overcoming the inefficiencies of browser-based DNS caching. Our implementation adds a minimal set of new flow rules to OpenFlow switches, and the NFV IP-rewriter portion by design can easily be virtualised and horizontally scaled. This makes it possible to implement our proposed solution in most pre-existing OpenFlow networks. As such, our architecture allows for network operators to agilely detect and implement network changes that can immediately affect their end users. By providing this ability,

[8]For example, in scenario B: Network Link Congestion, the amount of congestion the link is under, or how much bandwidth is available to each user.

we also enable ISP CDNs towards rapidly provisioning cloud services to create extra CDN replica servers on demand, I.E in response to a flash flood event. Additionally, we have directly demonstrated the shortcomings of current DNS caching schemes, thus providing a benchmark for future developments and research in this area. Our own future work in this domain looks towards implementing a redirection scheme utilizing anycast techniques within our proposed architecture.

ACKNOWLEDGMENTS

This project was funded in part by InternetNZ. Many thanks to Charles, Sam, Chris, and the rest of the staff at REANNZ for their help and inspiration throughout the project.

REFERENCES

[1] B. Frank *et al.*, "Pushing cdn-isp collaboration to the limit," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 35–44, 2013.

[2] J. S. Otto, M. A. Sánchez, J. P. Rula, and F. E. Bustamante, "Content delivery and the natural evolution of dns: Remote dns trends, performance issues and alternative solutions," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 523–536. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398831

[3] "Host resolution in chromium," Chromium devblog, 2012.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[5] "Ryu sdn controller framework, project website available at: http://osrg.github.io/ryu/," http://osrg.github.io/ryu/.

[6] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000. [Online]. Available: http://doi.acm.org/10.1145/354871.354874

[7] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1842733.1842736

[8] "The content delivery network (cdn): Deliverying the ultimate web experience," White paper, Bell Canada, 2010.

[9] A. Pathan and R. Buyya, "A taxonomy and survey of content delivery networks,," Technical Report, GRIDS-TR-2007-4, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia., Feb. 2007. [Online]. Available: http://www.gridbus.org/reports/CDN-Taxonomy.pdf

[10] "Load balancing without load balancers," Industry blog, CloudFlare, 2013, accessible at: http://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/.

[11] Y. E. J. Y. C. D. Jingguo Ge, Yulei Wu, *Investigation of Anycast implementation in Software-Defined Networking*. CRC Press, 2014.

[12] P. Georgopoulos, M. Broadbent, B. Plattner, and N. Race, "Cache as a service," in *23rd International Conference on Computer Communications and Networks (ICCCN 2014)*. IEEE, 2014.

[13] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342359

[14] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 459–473. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616491

[15] "Browser statistics," Browser statistics, w3schools, 2015, accessible at: http://www.w3schools.com/browsers/browsers_stats.asp.

[16] Y. Liu, S. O. Amin, and L. Wang, "Efficient fib caching using minimal non-overlapping prefixes," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 14–21, Jan. 2013. [Online]. Available: http://doi.acm.org/10.1145/2427036.2427039

[17] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466