**UNIVERSITY OF NAIROBI**

**COLLEGE OF BIOLOGICAL & PHYSICAL SCIENCES**

**SCHOOL OF COMPUTING & INFORMATICS**

# END USER REQUEST REDIRECTION PERFORMANCE IN CONTENT DEVELOPMENT NETWORK USING SOFTWARE DEFINED NETWORKING -BASED NETWORK

**BY**

**ALEX MUNYOLE LUVEMBE**

**(P53/79207/2015)**

**SUPERVISOR**

**MR. ERICK AYIENGA**

---

**A RESEARCH PROJECT REPORT SUBMITTED TO THE SCHOOL OF COMPUTING AND INFORMATICS IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF MASTER OF SCIENCE DEGREE IN DISTRIBUTED COMPUTING TECHNOLOGY OF THE UNIVERSITY OF NAIROBI.**

**NOVEMBER 2016**

## DECLARATION

This project is my original work and, to the best of my knowledge, this research work has not been submitted for any other award in any University.

Alex Munyole Luvembe ______________________________ Date: ________________

**(P53/79207/2015)**

This project report has been submitted in partial fulfillment of the requirements for the Master of Science Degree in Distributed Computing Technology of the University of Nairobi with my approval as the University supervisor.

Sign: ______________________________ Date: ________________

**Mr. Erick Ayienga**

**School of Computing and Informatics, University of Nairobi**

## ACKNOWLEDGEMENTS

I thank the almighty God for this far he has enabled me. I would like to give gratitude to my lecturers and fellow students for the support they accorded me. I wish to express my sincere gratitude and thanks to my supervisor Mr. Erick M. Ayienga for his support and guidance that enabled me to complete this research project. I also recognize and appreciate constructive criticism of the panel team; Dr. Elisha Abande, Dr. Mwaura, Mr. Erick Ayienga and Prof. Omwenga.  Your feedback greatly helped me during my project progress.

My colleagues at work for sacrifices you made and support you gave, I am very thankful. I also thank my sisters Harriet, Maryanne and Caroline for great sacrifices, support and prayers.

My parents, Mr. and Mrs. Bernard Muchera Lubembe I thank you for being inspirational to my life. Your relentless support as brought me this far!

## TABLE OF CONTENTS

## ABSTRACT

Content Delivery Networks has created a sharp rise of internet traffic in recent years. New technologies has simplified users access to rich content, however, as the internet continues to grow and more users access content stored in CDNs, issues such as reduced scalability, reliability and availability becomes a challenge to end users. This study investigates end user request redirection performance in content development network using software defined networking - based network. The results should assist researchers in network management, Internet Service Providers (ISPs) in managing request redirection as a service. An experiment to determine effects of client-side DNS caching on the contemporary DNS-based redirection technique found out that both Firefox and Chrome are prone to client side DNS caching. When subjected to a passive testing, Firefox and Chrome "timeout". Moreover, users had to refresh their browsers for over a minute to get a response from a CDN server. The findings further indicate that Software Defined Networking improves users' quality of service in request redirection. It took ten seconds for multiple users accessing CDN servers to download content and get redirected. Further, the Domain Name System augmented with Network Address Translation technique, based on SDN approach is able to reliably redirect user request to an optimal CDN server. This is when if it discovers the DNS contains out of date information. The ryu controller gathered network statistics and pushed SNAT and DNAT rules. The rules directed TCP packets, hence reliably redirecting users to a better performing CDN server.

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF ABBREVIATION

| | | |
|---|---|---|
| HTTP | - | Hypertext Transfer Protocol |
| CDN | – | Content Delivery Network |
| SDN | - | Software Defined Networking |
| DNS | - | Domain Naming System |
| TTL | - | Short-to-live |
| PECA | - | Portable Extended Cache Approach |
| RTT | - | round trip time |
| ALTO | - | Application Layer Traffic Optimization |
| POF | - | Protocol Oblivious Forwarding |
| PRP | - | Parallel Redundancy Protocol |
| QoE | - | Quality of Experience |
| NAT | - | Network Address Translation |
| DNS | - | Domain Name System |
| ISP | - | Internet Service Providers |
| IP | - | Internet Protocol |
| RTT | - | Round Trip Times |
| PECA | - | Portable Extended Cache Approach (PECA) |
| TCP | - | Transmission Control Protocol |
| GENI | - | Global Environment for Network Innovations |
| URL | - | Uniform Resource Locator |

## DEFINITIONS OF TERMS

| | | |
|---|---|---|
| Content Distribution Networks | - | Businesses that provide online content services |
| Hyper Text Transfer Protocol | - | It is standard protocol available for data communication on the internet |
| Network Address Translation | - | A process of remapping a single IP address into the other by modifying network information available in Internet Protocol |
| Load Balancers | - | Software or hardware used to distribute load between servers equally thus increasing server performance |
| SDN based technology | - | It is a technique that involves separation of the network control plane physically from the forwarding plane and where the control plane controls several devices. |
| DNS Caching | - | It is database maintained by the computer system temporary and may contain all the sites a user has visited. |
| Uniform Resource Locator | - | It refers to an address of a resource available on the internet |
| IP anycast | - | It is a model where the same IP prefix is made available from several locations |
| Latency | - | In Networks, latency refers to the amount of time a packet of data hops from one point to the other |
| OpenFlow | - | It is an experimental protocol enabling researchers to setup and experiment with campus networks |
| Flash Flood | - | Refers to sudden increase or surge of users accessing a particular CDN |

# CHAPTER ONE

## 1.0 Introduction

The design of Internet architectures traditionally targeted static content. Static content was typically network intensive. They were suited for technologies such as web caching, replicating information on mirrored servers as well as Hyper Text Transfer Protocol (HTTP). Similarly, the use of the Internet by individuals and organizations reflects the dependency of the web. Thus, users accessing information as well as other content online services have led to huge traffic generated. Such massive increase in traffic has resulted in better alternatives of improving the performance and speed of the internet. It is only through such increased speed that user request redirection becomes palatable to the end user. However, with increasing demand for rich multimedia content, a surge in internet traffic content, and higher availability bandwidth; the applicability of the alternative designs needs to guarantee desired performance for user requests (Harahap & Wijekoon 2013).

Today, Content Distribution Networks (CDNs) have emerged as a result of offering scalable, yet prompt response time to users' requests. Depending on a user's location and server, CDNs render content promptly to the requesting user. The dynamic nature of CDNs has, therefore, generated interest among researchers. Many investigators have recently explored the field to enlist standards guaranteeing maximum user request performance. According to Garg (2015), these solutions are effective in improving performance on static content than on dynamic content. However, several solutions proposed have received a fair share of criticism. One of the major cited shortcomings of these approaches revolves around the assumption they fail to maintain consistency. Replication of information on mirrored servers may redirect user request improving the traffic performance through selective replication.

Some studies have proposed algorithms to address the issue of request redirection in CDNs. The algorithms proposed were a suite of algorithms that could optimize the response time of dynamic contents on the web while reducing resource use. Despite being a success, such ambitious approaches are flawed because they tend to overlook redirection request on static content which also contributes to a considerable percentage of traffic in CDNs. Ranjan & Knightly (2008) similarly, cite that hostnames for websites hosted by CDNS currently are resolved using DNS. The DNS service offers end users with an IP address that map a particular user's request to an optimal CDN node at a given time. Though this is important for the user, cached DNS information may contribute to users accessing a non-optimal CDN server.

Unlike other studies, this study aimed at investigating end user request redirection performance in CDNs using an SDN-based network. Thus, this research investigated the effects of client-side DNS caching on the contemporary DNS-based redirection technique. The research also, designed and implemented a user request redirection using Software Defined Networking (SDN). This study used DNS approach, augmented with Network Address Translation (NAT). NAT had the capability of bypassing client-side DNS caching when used in user request redirection, hence, improving user request redirection performance in CDNs.

## 1.1 Background to the Study

The rapid growth of the internet in the recent past has contributed to the rise of Content Delivery Networks (CDNs) as the primary providers of online content. With Cisco's prediction by 2019, sixty-two percent (62%) of all global Internet traffic will cross CDNs compared to thirty-nine (39%) experienced in 2014, CDNs are progressively scaling their network footprints in tandem with the emerging traffic requirements (Cisco 2016). CDNs are being compelled to find better approaches of guaranteeing Quality of Experience for end-users. Most of the current CDNs seek to reliably and efficiently render optimal content emanating from content providers to end users. Though the operational principles of various CDNs are not the same, they all envisage a similar structure. Replicas containing content are moved from the service provider's server and placed in replicated servers spread within the CDN's network. Replication of the content on the CDN's network helps a user access desired content.

Several factors determine access to content placed on a replica server according to Triukose et al. (2011). They include latency, network load, geographical proximity and content availability. These factors are crucial to the content providers and the end user since they improve timelines delivery as well as load balancing. The proliferation of the dynamic content and the growth of the internet to accommodate novel services has become a major area of research by various scholars. Most researchers have concentrated their work on user request redirection as a service. Little research is available publicly that addresses user redirection performance in CDN using an SDN-based network. Calder et al. (2015) point out that coping up with user requests demands and render optimum performance; an end user request redirection should envisage several features. The redirection approach should be scalable, flexible, and transparent and should be able to decouple a server. A redirection service with these features improves performance by minimizing challenges in congested networks by making the resources reachable, improves scalability and maintains high accessibility of services.

Over the years, studies done in CDNs show that CDNs have evolved and addresses the growing needs for secure and efficient delivery of content online. The widely used request redirection used in CDNs is the Domain Naming System (DNS). The pervasiveness of DNS as a directory service makes it a suitable method. However, other studies report that DNS harbors weakness such as having multiple phases of redirection and not embracing scalability (Elkotob & Andersson 2012). This causes various stages of DNS redirections to occur because DNS caches miss causes the Round Trip Time (RTT) to be centralized on DNS servers. Similarly, the Short-to-Live (TTL) which is used to respond automatically to changes in the network environment increases DNS server load, thus, affecting redirection performance. Currently, an established working group in (IETF) is crafting a set of standards aimed at interconnecting CDNs. This standard is referred to as (CDNI). The team is using SDN approach to attain CDNI standards. However, the CDNI standard is focused on request redirection between downstream and upstream CDN providers in a CDN and not between a CDN replica node and an end user. HTTP request direction is also decisive in redirecting client application in CDN. In his research, Vasilakos et al. (2015) postulated that a web server alters the HTTP client header by modifying the IP address when the server receives the request (Baggio 2004). Despite being used on the internet for years, HTTP redirection is not a transparent service.

In his expose, Baggio (2004) observed that Domain Name System (DNS) is widely used by many CDN's. CDNs use DNS to route users' access requests to an optimal CDN replica server. This approach has been criticized by researchers. The existing mechanism for redirecting users overlooks the DNS performed by end users. This leads to effects such as underperformance and decreased browsing experience by end users as a result of DNS caches from optimal CDN nodes. In his studies, Baggio (2004) designed a distributed redirection scheme to enable users access the closest replica by exploiting locality. By using distributed redirection, the user request redirection scheme was undertaken at the client machine locally. However, the author did not perform evaluation and comparisons to ascertain the performance. Various researchers have also attempted to propose redirection techniques based on SDN to decentralize traffic as well as manage single points of failures during a redirection process. Handigol et al. (2010) proposed a system known as Aster*x. The technique aimed at improving performance of user requests redirection by reducing the network and server overheads. Aster*x achieved this goal customizing flow redirection common in arbitrary and amorphous networks.

Other studies have focused on new redirection approaches. Wang et al. (2011) devised an algorithm based on OpenFlow standard where network switches divide traffic over server replicas using packet handling rules. A separate controller installed these rules. He also introduced wildcard rules. For the controller to perform efficiently, it exploited the switch to aid wildcard rules to improve scalability meant for directing traffic of clients to replica servers. The algorithm introduced also computed algorithms that supported traffic distribution automatically. The algorithm simplified load balancing by maintaining existing connections. Although being a scalable solution, this technique has inherent shortcomings. The technique overloads the control plane of a switch. For example, a client requesting to reach its replica server gives priority to the initial call originating from the client to install forwarding rules before proceeding. Also, the solution overloads the switch with many rules.

In other researches, the role of load balancing in user request redirection has been extensively explored. Over the years, load balancers have been used in CDN data centers to enhance performance and guarantee the reliability of user request redirection. A load balancer typically directs each user request to a given replica. Load balancers use consistent hashing to achieve this aim (Calvin et al. 2016). However, this approach has limited customizability besides being an expensive piece of hardware. Studies have shown that a user request method should be inexpensive as well as less sophisticated to allow flexibility in managing request in simple ways. Besides, the solution should be able to scale, as the number of user requests and replica grows.

All in all, on examining published studies on the effects of DNS caching by end users, it can be seen that the effects of DNS caching by end users have not yet been considered adequately. Furthermore, as the Internet continues to grow; new technologies are emerging. Hence, these developments have a direct effect on the performance of services rendered by CDNs.

## 1.2. Statement of the problem

Researchers investigating end user request redirections in CDNs have shown that while multiple techniques exist to facilitate user request redirections in CDNs, none redirects user requests transparently. While DNS-based request redirection is widely favored because of its ubiquity to redirect user request to an optimal CDN, issues associated with stale information hampers its performance. According to Liang & Yang (2015), stale information emerges as a result of topology change and to when end users' cache DNS information which DNS is unable to navigate. Further, DNS user request redirection depends on the end users making DNS request so as to be provided with an IP address (Liu et al. 2012). The IP address granted maps users to optimal CDN replica servers. However, studies have shown that though this process is better in practice, it is unable to consider cached DNS information from end users. This aspect becomes a major issue if multiple CDN replicas are made available during the meantime. Although a majority of CDNs assign low TTLs to their DNS entries to work around this problem, this allocation is not honored. This is the reason why this research investigated end user request redirection performance in content development network (CDN) using software defined networking (SDN)-based network.

## 1.3. Objectives of the Study

The general aim of this study is to investigate end user request redirection performance in content development network (CDN) using software defined networking (SDN)-based network.

### 1.3.1. Specific objectives

Specific objectives:

i. To investigate effects of client-side DNS caching on the contemporary DNS-based redirection technique

ii. To investigate SDN-based technique in enhancing quality of service in user request redirection

iii. To design a user request redirection method using SDN that can redirect users reliably when it recognizes DNS contains out of date information.

iv. To evaluate the effectiveness of the proposed user request redirection technique in improving end user request redirection performance.

## 1.4. Significance of the Study

This study is vital to researchers in network management, Internet Service Providers (ISPs) as well as CDNs, especially in managing request redirection as a service. This is because the web is increasingly growing with aspects such as video streaming that render web video to the TV, and other endpoint devices, file downloads, social networks and global web portals among others. These phenomena will require an adequate understanding of request redirection to prevent network congestions while at the same time improving user browsing experience. Thus, this study will provide these stakeholders with an understanding of appropriate techniques of routing content in a CDN to improve performance while maximizing idle resources available, for instance, the bandwidth management.

In addition to the literature on request redirection, the study also contributes to two broad areas of research that are SDN technology and CDNs. This study will contribute to other studies by designing an approach based on SDN technology. The proposed approach will redirect requests optimally. Overall, this research will contribute to the body of knowledge in the area of end user request redirection performance in CDN networks.

# CHAPTER TWO
# LITERATURE REVIEW

## 2.0. Introduction

Much literature exists on end user request redirection performance in content development network (CDN) using software defined networking (SDN). This study focused on studying end user request redirection performance in CDN using SDN. Literature review began by searching professional journals in leading e-libraries. There were an overwhelming number of papers written on user request redirection. The literature review on end user request redirection performance concentrated on techniques available for end user request redirection performance in CDN networks, SDN-based techniques in enhancing the quality of service in user redirection, effects of client-side DNS caching on the contemporary DNS-based redirection technique and the SDN. The literature review also revealed gap present in prior studies as well as demonstrated the relevance of the current study.

## 2.1 End-User Request Redirection Performance in CDNs

Various studies have investigated end user request redirection performance in CDNs using different approaches. Scholarship available in the area has classified user request redirection performance in different categories. Broad categories available include end user request redirection using load balancers, DNS and collecting information to aid request redirection.

### 2.1.1 End-User Request Redirection Performance using Load Balancers

The rise of rich content on the internet as a result of CDNs in the last 20 years has attracted extensive discussion in practice and research. Traditionally, users were redirected to specific content using load balancers; however, a major rise and demand for ultra high-quality media and real-time videos have created a challenge on load balancers in redirecting users request optimally. Load balancers distribute load between servers and reduce response time when multiple user's requests are accessing servers. Load balancers focus in distributing requests from users across the entire CDN. The inability of the load balancers being unaware of bandwidth required by the users also hampers end user's requests being redirected at a slow pace, increasing response time (Calin & Schulzrinne 2015). Moreover, the current technology inhibits traditional load balancers from fulfilling clients bandwidth needs.

Recently, in an attempt to blend immense, and divergence research findings that have emerged in this discipline, Chen et al. (2016) proposed and tested an algorithm as an improvement of existing load balancers. Using the algorithm, they found a variant in user request redirection and reduced response time than existing approaches. The new model used SDN technology. The algorithm focused on minimizing server load while improving server response time based on the request directed to it. While the algorithms used in load balancers such as Least Load and Round Robin are useful in balancing loads, they harbor some weaknesses. For example, although Round Robin is straightforward and fast, it is poor in precision when balancing load requests with varying complexity and sizes. On the other hand, the Least Load algorithm only considers a load at a particular server before distributing the load to the server. This phenomenon raises a possibility of leaving the systems idle, decreasing the scheme's efficiency.

Saini (2015) presented a hybrid model by merging Static Load Balancing and Dynamic Load-balancing to improve the performance of user's requests. A load balancer introduced inherited the properties of dynamic and static load balancing and overcame the limitation of both the algorithms. Though the hybrid model proposed was successful in balancing the load and improving user's request performance, it was unable to illustrate the criteria used for selecting Round Robin and Least Load algorithms and leaving out the random property as another property with traditional load balancers.

Extant studies in request redirection in CDN's literature have also suggested different components be used with load balancers, rather than using a single element. Multiple components strengthen the probability of increased response time and reliability using redundancy. The load balancer has the capability of distributing user requests among other servers. Saini (2015) shows that one of the most common methods for load balancing is using a DNS server. A DNS server provides clients with IP addresses to access content located at a particular site. However, the weakness of a DNS server is the name to IP mappings. Name to IP mapping may result from a DNS lookup. DNS lookup has the potential of caching contents present in the flow route of the server and the client. As a result, load imbalance may occur because requests from a client may bypass the DNS server and aim at the server directly, increasing response time. Moreover, a client request with a higher priority would be granted a name to IP address mapping with a relatively shorter lifetime assigned, with a low client request rate.

While prior studies are significant to this study, they differ in some ways. Some propose algorithms to be used by load balancers while others designed hybrid models to enhance end user's requests. However, this study departs from previous studies. It investigated end user request redirection performance in content development network using an SDN-based network. The study used Domain Name System augmented (DNS) with Network Address Translation (NAT) Technique.

### 2.1.2. End-User Request Redirection Performance using Domain Name System

DNS is employed in CDN networks because of its directory service ubiquity, transparency and simplicity. However, several studies have demonstrated the weaknesses of DNS in supporting the performance of user requests. Studies have classified DNS shortcomings like lack of dependability, performing resolution at the domain level, dependency on Internet Service Providers (ISPs) and reduction of TTL (Garg 2015). DNS request redirection does not redirect client request dependably to the nearest replica. On the other hand reducing TTL for DNS records escalates the weight on DNS servers through aspects such as magnitude. Furthermore, claim that the assumptions of proximity between DNS server and clients may not often hold. The assumptions lead to major ineffective request routing judgment (Xiang et al. 2015).

A redirection model should perform request direction per item level. However, DNS does not achieve this goal. DNS redirection does not scale well and entails multiple hierarchies of redirections. It means that in a cache miss, subsequent lookup has to take a round trip time to a centralized server. The results of DNS resolutions are locally cached in the DNS servers for later use. Hence, to permit quick adaptation to evolving servers and current network request routing, DNS servers mark their query responses with a short-to-live (TTL) values (Akyildiz et al. 2014). This phenomenon increases major requests to DNS servers. Furthermore, DNS systems encounter scaling difficulty about supporting multiple content providers and a massive number of content providers. Similarly, DNS requests rely on Internet Service Providers (ISPs) DNS servers to send end- user's requests. In this case, the redirection server possess only the information of the IP of the client DNS server locally available, and not the correct client IP address in assessing client-server proximity. This process results in assumptions that clients are closer to the enquiring server which is not the case. Some of these challenges in DNS request routing have pushed a research a notch higher (Nygren et al. 2010) . They have an impact on user request response time.

Liu et al. (2012) proposed a modified DNS scheme, where the request redirection decision is pushed closer to the clients using DNS servers present locally. By this method, the DNS redirection to the DNS server returns a possible list of servers together with their valid load information. The local server then acquires extra information such as hop count towards other servers using the local Internet gateway router, network latency, and route bandwidth.

Literature explored on end user request redirection performance using DNS informed this study in two different ways. A redirection mechanism modeled should be transparent and independent. It should dynamically make redirection decisions based on the information available in the network to redirect user request optimally. Prior studies have concentrated in modifying DNS scheme to improve end user redirection. However, this study used DNS technique augmented with Network Address Translation (NAT). NAT's role was to conduct a destination address on the outgoing packets and redirecting users to an optimal CDN. This approach guaranteed transparency in end user requests redirection is improving performance in content access.

### 2.1.3 End User Request Redirection Performance using Collected Information

Request redirection involves gathering information regarding the state of the network and making an appropriate decision based on this information. Major studies in this area have concentrated on collecting information to assist in making routing decisions rather than on ways of performing redirections. Studies have involved the design of systems to monitor connections statistics, incorporating clients in computing measurements to framework concepts used by load balancers. Whitaker et al. (2002) explored ways in which request redirection could be improved in a CDN. He proposed a system known as a Webmapper. The system was designed and co-located with CDN servers. The designed system passively monitored client connections statistics. The statistics included connection Round Trip Times (RTT) and client addresses space. The information collected was then used to make routing decisions and aggregate client address spaces in liaison with DNS request routing system.

Other researchers such Mukhtar & Rosberg (2003) also contributed significantly in this area. They proposed active client side involvement in measurement; in this model, immediately the client connects to a server, the request redirection system in the server pushes a probing script (usually written in JavaScript) into the client's web browser. The JavaScript mutely instructs the client to contact and compute loading times of the servers and the report the results. The collected information is used by building the system's request routing database. Similarly, in improving performance for request redirection directed to web servers, Juneja & Garg (2012) devised an ANT framework for load balancing. The framework concept was to have a collection of agents that can individually perform actions such as rejection or selection of hyperlinks for calculating the load, navigation and sum these individual measures in a universal substrate. The authors projected activeness, both at the server and the client side. The server side's role was to reply the request whereas the client's side was responsible for generating user requests.

Most prior studies in the area of collecting information to aid in request redirection have proposed systems to monitor connections statistics in a CDN network. Other studies have proposed frameworks regarding gathering information to improve request redirection performance. However, this study departed from prior studies. The study used the SDN technique to gather network information. OpenFlow was driven by OpenFlow controller which was responsible for collecting network information and pushing the same information to the switches based on rules inserted in the switch. The rules guided the redirection process to an optimal CDN server by pushing SNAT and DNAT to the network switches.

### 2.1.4 End-User Request Redirection Performance using other Approaches

There has also been several works highlighting mechanism to achieve end user request redirection using other techniques. Frank et al. (2013) sought to demonstrate the working of NetPaaS in a user-server assignment in a CDN. The NetPaaS consisted of three components; the server allocation interface, network monitor, and the informed user task. The model showed that when CDN receives a DNS request, it uses internal information to service the request. The server selected by the DNS request was determined by the location of where the request originated. When CDN makes a choice in choosing servers to accomplish a redirection, it sent to a resolver. The resolver again sends the request which in turn sends it back to where it has originated from. However, in a circumstance the content being sought is unavailable locally; the server is bestowed the responsibility of fetching the content from any other server or the original server. CDN's using this mechanism gives authority to CDN for DNS resolution for the domains for contents delivered via CDN. Every CDN is granted authority for some domains which satisfy the DNS requests. The first action which a client executes on a given content is the DNS name resolution. In this case, a DNS server first points to a something unique to the request routing mechanism. For example, the DNS can return a suitable surrogate in a CDN or, it can redirect the privilege to idle and available domain level to another request redirection DNS server belonging to another DNS.

Miura and Yamamoto (2002) also came up with a network layer anycast using existing networking paradigm. In this approach, network routers are improved with application layer functionalities. The functions permit retrieval of traffic and monitoring content requests to measure the application round trip time (RTT). Through this approach, the authors posit that by using application layer RTT than network RTT, the server and the network connected latencies are considered. The issue of latency leads to a better user-perceived performance. Xiang et al. (2015) proposed a peer-to-peer architecture. Peer-to-peer architecture embraced the Tapestry (peer-to-peer architecture) among the

CDN servers. End users connect to the local and nearest server and request contents which are passed from one server to the other until an appropriate replica are found.

Other approaches available on end user request redirection have placed emphasis on improving existing anycast network layers and proposing models to aid in request redirection. However, other than proposing a request redirection design using SDN-based technology, this study evaluated the proposed solution to determine its performance in redirecting users.

### 2.2 User Request Redirection Approaches in Content Development Network

Request redirection approaches have been used widely as a conduit of extending the scale and reach of CDNs. Giancarlo (2012) showed that currently, there are multiple request redirection approaches available. They are classified under the application-layer; transport-layer and DNS-based request redirection. A redirection system entails a standard of policies in its operations in redirecting users to replica server's optimal servers. Such standards encompass content availability, available bandwidth, the load of the surrogate and network proximity.

#### 2.2.1 DNS-Based Request Redirection Approach

Prior literature reveals that the ubiquity of DNS makes it widely accepted as a request redirection method in CDNs. A specialized DNS server is used to allow DNS carry out DNS resolution process. The dedicated server can return different categories of CNAME, NS or A records by analyzing user's metrics or policies. A DNS has several features which make it suited in request redirection. According to Aggarwal et al. (2007), one major feature of DNS-based request redirection is the single reply. A single response grants a DNS server authority over sub-domain or domain. In this approach, A DNS returns a favorable address of the optimal replica server in a recording to a requesting DNS server. The second feature is the multiple replies. DNS server uses multiple responses for several surrogates. The sequence in which records are dispensed back is taken as a concept to redirect several clients by embracing a distinct client site DNS server. Other features standards with DNS-based request redirection are Multi-level resolutions, NS redirection, and CNAME redirection.

Xu et al. (2015) proposed a routing model where the routing decision was pushed near the client by using a customized DNS local server. This model permitted the CDNS request-routing DNS server to return a listing of all probable servers with locally available DNS servers, and with current load information. The locally available DNS server then acquired additional information such as route bandwidth, network latency and hop count about a candidate server using local Internet gateway router. The information was kept and utilized when a local decision concerns better performance in network bandwidth balancing and access latency in comparison to secure schemes being used in commercial CDN is still not clear. Moreover, this model would need substantial modification to the user's side DNS servers. This process is usually beyond the scope of CDN providers. According to Baggio (1999), DNS facilitates resolution at a domain level. An optimal request redirection approach should be achieved at an object level. Consequently, DNS request redirection directed towards servers are required to retort the DNS entries with short–to-live standards. This is needed to make them dynamically adjust in times of overloads or congestions. However, entries with STL such aspects augment requests directed to DNS servers

**2.2.1.1 Why DNS Caching?**

In 2014, Kende (2014) released statistics on browser usage around the world. On these browsers, it was observed that Chrome caches DNS entries for 60 seconds, Firefox follows with 60 seconds, Internet Explorer at 30 minutes while Opera caches DNS entries for 10 minutes. Therefore, it is imperative to infer majority of DNS requests for content hosted in CDNs have responses cached. Shravani (2014) observes that if additional DNS caches are being used, the issue is amplified further.

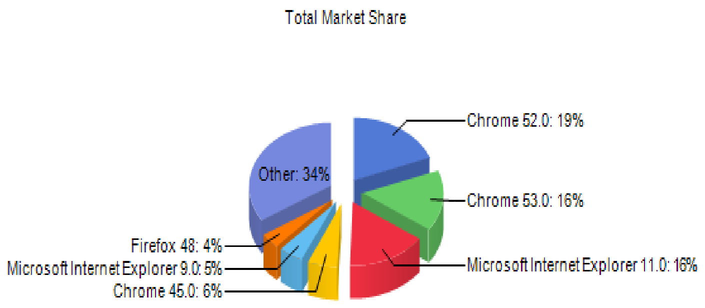


Fig 2.1: Browser Global Market Share

Internet Explorer, Firefox, and Chrome perform "link pre-fetching" (Acker et al. 2016). Link pre-fetching is a process in which a DNS request is made before a user follows a link. Browsers attain this aspect by "looking" at a page and performing DNS request for their hostname. Browsers also perform DNS request during startup to reduce the load time of the websites which the users accesses frequently. Link pre-fetching, and DNS caching establishes a scenario where an IP address of a hostname is likely to change while the end user still on the "old" IP address. This scenario may lead to poor performance or QoE degradation for the end user because content accessed may be stale or CDN server may congest. Mein (2012) observed that because of the challenges exhibited by DNS, a CDN redirection approach needs to incorporate link pre-fetching and end user DNS caching. If these elements are avoided, then any modifications to the CDN is likely to take more than a minute for its effects to be felt by all users of a CDN.

Georgopoulos et al. (2014) implemented an OpenCache. OpenCache was an SDN caching platform, still under research. However, the approach is more focused on providing a caching solution for ultra-high quality streaming video. The authors argued that OpenCache system can be used to improve performance. They implemented their solution using a flow granularity at an IP level. However, in this study, the end user request redirection will be based on TCP sessions. For several years, web caching has been widely embraced by the Internet Service Providers to minimize bandwidth, improve the content and user request performance on the internet. Besides, the technique has been used to improve reliability, traffic reduction and performance on the internet. A local cache serves a cached content more quickly even when the originating server's network is congested. However, maintaining cache consistency across the internet is a major challenge. Though consistency is less important with static contents, it becomes a severe issue with dynamic contents which at times may render expired content.

In caching, end users recently accessed web contents are stored in the cache memory of proxy servers. Subsequent request of the same nature is serviced using proxy servers reducing response time, network traffic and load on the server. Juneja & Garg (2012) came up with Portable Extended Cache Approach (PECA) to keep recurrent end user data in an unlimited cache memory. The proposed solution was done augment the computational performance of web service. Caching at the client side is synonymous with data scheduling. It improves the performance of information in dispatch systems in real time. An efficient caching system retains critical data users will likely access and minimizes requests attempts directed to the server. This approach ultimately saves bandwidth and reduces server's workload. Furthermore, it reduces access latency by servicing local requests with cached data at the clients.

### 2.2.2 Transport Layer Request Redirection

Major research in CDN request redirection has exploited transport layer in an attempt to enhance the performance of end-user requests. Scholarship in this area affirms that request redirection at the layer of carriage offers finer levels of granularity because client's requests attract closer inspections. A request redirection approach inspects the client's packet first information requests to decide the selection of a surrogate. The inspections provide rich data concerning the client's layer four protocols, IP address, and the port information. The information extracted is combined with other metrics and user-defined policies to service the request.

Vaughan-Nichols (2011) shows that the overhead connected with transport layer redirection approach enable long lived sessions such as RTSP and FTP to be executed. However, the approach has the probability of directing clients away from congested surrogates. In most cases, scholarship has shown that the layer transport redirection can be used together with DNS techniques. The DNS approach can be used to make a decision on the optimal surrogate with favorable refinement achieved via the transport layer redirection. Shaikh et al. (2001) proposed request routing agents that he called "Smart Boxes." The Smart Boxes were deployed in diversified geographical locations and assigned similar any case address space. The content request destined for anycast address was transparently redirected to the nearest routing agent. The agent forwarded it to an appropriate server depending on proximity and relevant information found on the server.

### 2.2.3 Application Layer Request Redirection

According to Giancarlo Fortino (2012), existing applications embrace various approaches for redirecting connections dynamically. These approaches include network packet rewriting, application layer protocols and name resolution manipulation. These approaches differ depending on the clients' degree of scalability, responsiveness, transparency and dynamic conditions. An application layer request redirection examines a client's packet at a deeper level; that is beyond the transport layer header. A more in-depth analysis of a client packet necessitates a finer grain request redirection to an individual object. The approach is flexible. The process is undertaken in real-time, when the end user requests certain content. The client's IP and the granularity of information connected with the content being requested makes the application layer technique render efficient control of requests selections of the most optimal surrogate. According to Gupta & Kumar (2014) application protocols such as RTSP, SSL and

HTTP provides an appropriate condition how the session emanating from end users should be directed. A URL may provide favorable conditions of rendering specific contents or other services, for example, the Cookies.

Katabi and Wroclawski (2000) also modeled the IP anycast. This model involved assigning diverse host's common/similar anycast IP address. The packets send using this IP is transparently forwarded to the nearby anycast host. In this case, nearness refers to the routing metrics defined by the routing procedure. Harahap & Wijekoon (2013) compared DNS and router based redirection approaches using Service-oriented Router (SoR) architecture. SoR was a router capable of redirecting requests from the clients to a server with minimal costs (server processing time, server loads and network delay among other factors). While using SoR model, several factors were considered. The first factor was a given as the workload. Workload, in this regard, was a redirection probability expression. It was derived to increase performance and enable the router redirect request to the most optimal server. Also, the study determined the average response time that encompassed network delay and service rate. All these elements were performed to evaluate the most favorable redirection approach.

Average response time of requests was determined by varying parameters such as service rate, network delay, and server load. The researchers used PlanetLab network as a test environment. In the experimental setup, network delay of client 1 and SoR router was 0.5 averagely, as carried out in PlanetLab. Average delay from SoR was 30 ms which was taken using DNS take record. Network delay between surrogate servers to SoR was 36 ms. the packet was set to a range of 1500 bytes. The router recorded a service rate of 10 Mbps with 830 per second; SoR 5 Mbps with 415 packets per second; DNS server 10 Mbps with 830 packets per second and surrogate server 1 Mbps with 83 packets per second. The findings of the study showed that router redirection performed better. It recorded 50% service rate. Also, the SoR indicated a higher redirection possibility when sandwiched between two servers. This was because the SoR recorded 23.3% when compared with the DNS-based redirection (Harahap & Wijekoon 2013).

Akamai also designed a DNS-based request router that would direct end user to an optimal available CDN node (Akamai 2016). The router combined network topology and operator media server information with latency, load, and information provided by hypercache nodes to direct user requests. The router was successful for improving deployment of operator multi-screen services which was confirmed in terms of performance, scalability and improved QoE. The router worked by using enhanced DNS-based mechanism. The router translated host names of end user requests and mapped them to IP addresses of edge caches. The router also handled local and global balancing at all CDN levels. However, the router main strength depended on CDN cache to maximize traffic redirection performance.

## 2.3 Measurements

A measurement criterion provides a new, efficient and flexible approach in making use of redirection resources. Their importance lies in improving end user request performance and achieves pervasive geographical coverage. These elements are recognized by deploying an appropriate request redirection policy. Mostly, such metrics are used by request redirection systems to determine a favorable surrogate that can service end user's requests. Cain et al. (2002) point out that most metrics used are based on surrogate's feedback and network measurements. For achieving optimum results, multiple metrics can be amalgamated using feedback and proximity of surrogates to improve surrogate selection.

### 2.3.1 Proximity Measurement

One major metric used by request redirection mechanism is the proximity measurement. Proximity is used to redirect end users to "closest" surrogate. Proximity measurement in DNS redirection is directed to the local DNS server serving the end user's requests. Precise proximity of the client is obtained when its IP is accessed. Calder et al. (2015) point out those proximity metrics are exchanged between requesting entity and surrogates. Widely used measurements for proximity are "one way". This means "one way" reverse or acts as forward path for the packets coming from surrogates and directed towards the entity requesting. Proximity measurements are critical because the internet has many asymmetric paths. Further, Gupta & Kumar (2014) illustrates that a set of proximity measurements are obtained through active network probing approaches or embracing passive measurements techniques. When using active probing, a possible or past requesting entity is probed. The probing can either use one or more methods to extract more than one measurement from surrogates. A classic example is employing ICMP ECHO sent periodically from individual surrogate to a probable requesting entity Wai, Pai & Peterson (2012)

In their study, Calder et al. (2015) used the load, locality, and proximity to evaluate various redirection requests using the algorithm they had created. The authors employed large scale and detailed simulations to evaluate the various strategies they had chosen. The finding showed that the algorithm proposed recorded 60-91% improvement in scaling the number of servers as well as user perceived response latency was kept low. However, the weakness of the comparison was that more time and resources were required for evaluation. Also, they planned to deploy they proposed algorithm on a testbed and explored more implementation issues.

According to Frank et al. (2012), probing actively requires potential entities. The entities can be sourced dynamically. Based on active probing the entities performing a request requisition cache the requests as they come for later probing. However, the limitation of active probing is that the metrics are periodical. Further, NATs and firewalls may block some probes during the process. In a passive measurement, a measurement is achieved during data transfer from or to surrogate servers. According to Buyya et al. (2008), a fitting example of a passive measurement is a loss of a packet emanating from a client and directed towards a surrogate server. The packet can be evaluated by the behavior of the TCP. Cain et al. (2002) also affirm that latency measurements are learned by evaluating the behavior of the TCP. However, a hindrance of this approach is that it is connected to bootstrapping technique. A good method should ensure each surrogate is tested separately to achieve accurate results.

#### 2.3.1.1 Metrics

Turrini (2004) illustrated several metrics which can be used to calculate proximity. One of the metric he pointed out was the latency. The latency uses the network latency measurements to analyze a surrogate that harbors the minimal delay to the requesting object. Latency measurements are extracted using active probing and passive methods. The second metric is packet loss. The packet loss measurement is mostly used as a selection metric (Weaver et al. 2011). A passive metric is used to extract lost TCP header packet information. An active probing periodically evaluates lost packet. The third metric is the hop counts. According to Safavi & Bastani (2015), proximit metrics can be extracted from a router’s hops directed towards a surrogate to service a requesting unit. The last one is the BGP information. BGP attributes such as BGP PATH, and MED are important in determining the length of a BGP distance. BGP information should obtain at each surrogates location/site for it to be used.

### 2.3.2 Surrogate Feedback

Pathan, Broberg & Buyya (2012) provided other key metrics used for determining the redirection they had proposed. The authors observed that factors such as high user perceived performance, scalability, transparency, cost, geographic load sharing and QoS metrics. In the experiment had designed, they nominated random, geolocation, and utility based redirections to evaluate their performance in a CDN. They devised an experiment which runs simultaneously for 48 hours at each client location. The experiment allowed them to localize peak times. They also created two test files of sizes 5 MB and 1KB deployed by CDN allocator module. These records were customized to maximize performance. Consequently, the authors deployed test files in cloud while integrating them with CDNs. Further, they listed utility, response time and throughput as the performance indices. Response time indicated the duration an end user accessed services while throughput referred to the download speed of the test file. Consequently, the utility, which referred to the servicing ability of the content, was 0 to 1.

In the experiment, an identical but ultra-high inconsistency of web access load was designed. The experiment time was set to 2 hours, and at each interval, user’s sessions were executed. At every session opened, a persistent HTTP directed to a CDN was established. A client generated a “GET” request to download each test file, with each client request timeout set to 30 seconds.

## 2.4 Software Defined Networking (SDN)

In the recent past, Software Defined Networking has emerged as a technology simplifying networking architecture. According to Chen-xiao & Ya-bin (2016), SDN embraces a physical separation of a control plane from the network forwarding plane. It also centralizes several networking control devices. Ooka et al. (2013) notes that SDN uses OpenFlow protocol to define communication between the switches and the SDN controller. Various studies have investigated user request directions using SDN technologies. These studies have broadly examined SDN and associated features. Studies on SDN have focused on the load balancer and the decisions on where to direct user’s traffic, wildcards, rules and the flow rules used by the controllers to forward incoming requests. A significant contribution in this area is the work of Handigol et al. (2009). His study modeled a load balancer known as Plug-n-Serve. Plug-n-Serve was used in an unstructured network. The proposed approach was modeled in a Mininet environment using simulation. The load balancer attempted to reduce the response rate by balancing network congestion and the servers. The proposed approach displayed real time load in the network. Plug-n-Serve used a controller controlled by software to balance network load and make decisions in directing user traffic. The model made it easier to add new servers to a cluster dynamically.

Most studies on request redirection using SDN have endeavored to address performance deficiencies encountered with initially proposed models (Puopolo et al. 2011). The initial models approach been viewed as a better in improving the performance of user requests. For example, though the Plug –n- Serve modeled by Handigol et al. (2010) was able to improve the performance of user request, it was limited in scope. Handigol et al. (2010) designed an Aster*x as an enhanced prototype of Plug-n-Serve model. Aster*x was tested in a Global Environment for Network Innovations (GENI), an SDN enabled test environment. Despite being successful, other researchers Yu et al. (2010) criticized the model by claiming that it worked by dynamically creating rules for outgoing packets.

OpenFlow offers a scalable characteristic that centralizes all data in the data plane. The features provided by OpenFlow enables the data plane to direct packets selectively using intermediate switches that stores forwarding rules (Fei 2014). These features are important in three different ways. The first significant point is that centralization decongests the network by allowing only resources in need to be used. Secondly, the rules can be applied to the data plane improving user request redirection and ultimately performance enhancements. Lastly, it enhances user's QoE on the internet. In maximizing these features, Yu et al. (2010) proposed a DIFANE approach. It was an efficient and scalable approach. The approach concentrated data by directing packets selectively in the data plane using rules pushed in the intermediate switches. The approach worked by relegating the controller using partitioned rules over the switches. The unique characteristics of this approach were wild card rules could be expressed on its data plane. These rules could perform simple tasks of matching the packets. DIFANE's architecture was modeled with a controller. The controller generated flow rules which were pushed to switches with high processing and memory capacity. The proposed approach distributed network rules to authoritative switches that redirected traffic in the available packet path. Besides, the model effectively handled wildcard standards and scaled fast in cases such as topology changes and policy changes. A further evaluation further showed that the approach was scalable on a network with many hosts and flow rules. The major weakness of Plug-n-Play and DIFANE was that rules were applied only to a single client.

Related studies have also explored OpenFlow algorithms that apply rules to multiple hosts rather than applying rules to individual clients. Wang et al. (2011) proposed an algorithm that applied flows to several hosts in a network. The algorithm directed hosts to CDN servers instead of flow rules being pushed to single clients. The algorithm used an external controller that enabled OpenFlow switch forward traffic using the installed. Wang also enhanced his model using Plug-n-Serve. This approach used network load to dynamically redirect network traffic to optimal servers. Despite various attempts by researchers to find an efficient and optimum user redirection approach, the solutions tested have not been a success. For example, Wang's et al. (2011) model introduced more overheads along by overloading a switch with many rules. Also, the process of clients sending connection to an external controller caused delay.

In this study, OpenFlow and SDN will be used to establish redirection rules. These rules will then be used to redirect end users to a better performing node in a CDN.

### 2.4.1. OpenFlow

OpenFlow is a breakthrough technology in networking. ONF (2016) postulates that it is an amalgamation of network devices enabling a packet flow in a network using an external controller. The external controller is a major component of this technology. OpenFlow creates network reliability and robustness by implementing a redundancy which is agile to react to changes. In circumstances where maximum redundancy or packet loss requirements are necessary, replication of information and resources may become the most appropriate method (Wytrębowicz et al. 2014). The capabilities of OpenFlow driven networks such as analyzing traffic statistics using software, dynamically updating forwarding information and testing of novel ideas and applications and propose new capabilities has contributed to the ease of the technology.

### 2.4.2 Performance in OpenFlow

Different researchers have investigated the performance of user request in OpenFlow. The findings have elicited mixed results regarding the performance of user requests. Handigol et al. (2009) designed a load balancer to enhance response time of user request. It worked by evaluating network congestion in real time. The algorithm used software that extracted real-time information from running controllers, servers, and network load to redirect the traffic. The proposed algorithm simplified detection of new servers when added in a cluster. The load balancer was hosted over PlanetLab nodes connected by to an SDN controlled network. The algorithm was remotely controlled by an Aster*x. The HTTP request was generated by clients from multiple network locations. Using a GUI driven interface, the system captured three aspects of the system. The first aspect was the overall condition of the system such as the load of the server and the network links. The GUI allowed the researchers to vary the requests arrival rate; work brought by each application basing on parameters such as load exerted on the CPU. Wang et al., (2011) also proposed a Plug-N-Play algorithm using an Openvswtich and a NOX, an OpenFlow controller platform. They deployed the topology in Mininet with three replica servers, two switches, and some clients. The objective of the experiment was to ascertain server's adaptability to new balancing weight and overheads during transitions. From the findings, the performance evaluation showed that the load balancer adapted to changes based on the load balancing policies as well as overhead during transitions.

Also, Binder et al. (2015) proposed a TCP handover model in OpenFlow to improve client redirection. The proposed used SDN features such as the capacity of the SDN to capture and redirect packets to the control plane for processing. Binder et al. (2015) proposed model used real time clients and servers in a CDN environment. In addition to redirecting the requests, the prototype was fully transparent to the clients, did not require extra DNS queries, did not require implementation of application level redirect mechanism and was fast in establishing sessions compared with application level redirect methods (Binder et al. 2015).

## 2.5 Analysis of Possible End-User Request Redirection Techniques

As the major area of this proposal was to design a request redirection approach, it was imperative also to describe the design goals that guaranteed an efficient and simpler implementation of the proposed solution. The proposed solution was implemented incrementally. This was to help reduce network performance as well as lower implementation costs. Also, the proposed solution scaled to accommodate network services. These requirements guided the design and the execution of the proposed solution.

### 2.5.1 IP Anycasting Approach

According to Juneja & Garg (2012), anycasting technique uses one- to- many associations of a hostname to CDN replica server endpoints. For example, when a browser places a DNS request for content hosted on a CDN, it receives an IP of an anycast address. The address provided directs the browser to the nearest replica server that responds to that address. This process diminishes the challenge caused by a DNS caching because the IP address available in an anycasting setup does not change. Calder et al. (2015) while analyzing the performance of an IP anycast in a CDN, they noted that IP anycast presents a mixed performance picture. By analyzing data extracted from Bing search service, they found out that though IP anycast performed well in redirecting user request to a nearby front end by 20%, it lacked control. However, they noted that despite its inefficiencies, IP anycast are stable, and a simple prediction scheme can be used to drive DNS redirection for end users undeserved by anycast. This strategy can improve the performance of the end users by 15% - 20%. In another study, Flavel et al. (2015) designed a FastRoute. FastRoute was scalable and operational anycast request redirection system to improve the performance of user redirection in accessing online services. FastRoute performed user request redirection by concentrating on proxy selection mechanism. The study noted that although FastRoute collocated DNS services and Proxies in the nodes location, it widely sacrificed the load instead of the arriving load on a Proxy server. Thus, the study concluded that FastRoute, an IP anycast technique is prone to user traffic overloading because the CDN does not have the capability of controlling which type of proxy receives traffic, leaving the process to the mercy of ISPs.

The nature of IP anycasting establishes a major effect of polluting the routing system globally. This is because it depends on the BGP to work. The tendency to pollute the routing system may often lead to network incidences that directly affect network uptime. A classic example of this effect was observed in 2014. According to Hutchison (2014), a CDN provider in Latin America accidentally directed traffic meant for CloudFlare to a data center. The traffic was so huge, and it overwhelmed the data center's capacity. Such failures often disrupt network services as well as affecting request redirection performance because the network uptime is reduced. As an alternative Hutchison (2014) illustrated that some CDNs have using anycast by embracing BGP routing instead of DNS redirection. However, relying on BGP postulates that network with anycast IP requires custom configurations and setup per network. Such configurations are complex and difficult to setup (Cicalese et al. 2015). Moreover, the emerging trend of using cloud services to aid CDNs to create replica servers is gaining much popularity. This trend can result in issues with IP anycasting. However, Calder et al. (2015) illustrate that because anycasting IPs directs requests to the local and nearest server that responds to anycasted IP address, it will make the established server to "receive" requests from nearby clients. Recent research by Schmidt et al. (2016) points out this behavior is present in current anycast systems. Further, while current CDN service providers that use IP anycast may have approaches to bypass this aspect of route flapping, the author is currently unaware, and thus, they will not be considered in light of this IP anycasting.

Currently, IP anycasting is not prone to client-side DNS caching; it is not affected by remote DNS servers. However, it is susceptible to route leakage, requires complex configurations, experiences failures, unable to use cloud services as a result of route flapping. Additionally, it is unable to use information such as congestions available in the network. Chen et al. (2015) point out instead of overcoming the challenge within DNS; CDN providers proposed an extension known as edns-client-subnet for DNS extension (Melorose et al. 2016). This is where a client IP address prefix is passed to the Local Recursive Resolver (LDNS) and later pushed to the authoritative name server. Recent studies have shown that some CDNs such as Akamai are using the extension. However, little is known about its performance improvement.

Trossen et al. (2016) explain that while the SDN-based technique of implementing IP anycast could solve some of the challenges such as the inability to make use of network information, other difficulties associated with IP anycasting outweighs benefits that IP anycasting provides. These challenges include complexity in configuration and not making use of cloud services to establish additional network resources. Such shortcomings make IP anycasting unsuitable for the expanding needs of CDNs (Clara et al. 2016).

### 2.5.2 Domain Name System augmented with Network Address Translation Technique

The request redirection technique proposed by this research used the current DNS based CDN redirection approach. However, it augmented it with NAT to bypass DNS caching at the client side. This section explored in detail why this approach was selected.

The most common technique of redirecting user requests in CDNs is augmented DNS servers. As mentioned earlier in this study, the limitation of DNS request redirection is clients perform DNS caching that does not embrace TTLs created by the CDNs DNS resolver. This scenario indicates a restriction in a way that users only perceive performance improvement of the rapidly expanding CDNs minutes after it has expanded. This also means that it takes a while to redirect users to other CDN replica nodes that exist in the network. Occurrences exhibited leads to windows where users experience reduced QoS of the network despite resources being available (Silvestre et al. 2013). A major aim of CDNs is to render dynamic rich media such as video on demand and live TV; low QoS attributes such as bitrate and access delay time significantly reduce user satisfaction by reducing CDN performance. A recent study commissioned by Ghasemi et al. (2016) pointed out that an increase of one percent in buffering ratio lowers users browsing experience engagement for more than three minutes while watching an online video lasting more than 90 minutes. As user expectation for better performance of the network continues to increase overtime, the performance of the network will become a major factor in user satisfaction.

Different scenarios may compel a CDN to redirect users rapidly away from a node they are currently connected to. One scenario is the flash floods. Flash floods become a major issue when there is a sudden influx of users. Flash floods have the likelihood of overloading a CDN cache location. Fei (2014) provided a classic example of this situation. HBO GO network experienced a flash flood when they were launching the Game of Thrones. Flashfloods caused performance issues affecting over a million users. Such disruptions are not conducive for business as it results in loss of trust on the network of channel reputation of a CDN. Performance issues as experienced by HBO GO could have been avoided by implementing a CDN request redirection mechanism that can accommodate a dynamic introduction of a new CDN replica node. Studies accomplished by Akamai (Chen et al. 2015) points out that it is viable to introduce, change and migrate online servers in unnoticeable time. Such studies confirm that a CDN network to overcome flash floods incidents by allocating extra CDN replica servers as demand increases.

Olteanu et al. (2015) also opine that it is possible for CDNs to outsource middlebox processing to the cloud. Further, they also affirm that cloud can allow redirection using multiple cloud PoPs by relying on a DNS-based approach that is related to its use ad function in CDNs. Also, other scenarios may include sudden link failure, the flash increase of current content and redistributing link congestions across the network (Sahri & Okamura 2014).

As the literature points out, it is vital to keep a high QoS for different categories of contents a CDN is likely to provide. Also, DNS caching postulates that any changes to a network topology may have a potential of taking over a minute for end users to be aware, therefore, it is efficient and desirable to have a request redirection technique that can overcome the effects of a DNS caching by end users. Noting that the current augmented DNS technique for request redirection is so pervasive and effective in redirecting end users to a CDN replica server, this proposal made a decision to design an approach where the approach would be ideal to augment the current DNS rerouting approaches in a way that avoids DNS caching. Thus, the proposal proposes a CDN request redirection approach where end user request redirection occurs via the conventional augmented DNS technique. However, if the network determines the end user is requesting content from a non-optimal node, the request is then redirected transparently to an optimal node for that user. This request redirection will happen at the edge router that serves that user.

## 2.6 Architectural Design

The study used Software Defined Networking technique to install flow rules in a switch. The rules installed were used to redirect user's packets to an edge router. The network consisted of an SDN controller with access to network information in a CDN network. The information gathered were used to determine if or not to push flow rules to edge switches to enable the switches to determine if they should perform a redirection on packets that pass through them. End user request redirection was established by determining destination NAT on the switches outgoing packets. The packets from the switch redirected users to an appropriate CDN replica server. The packets replying from a CDN replica server had a source NAT applied to them at the edge router before they were sent to the end user.

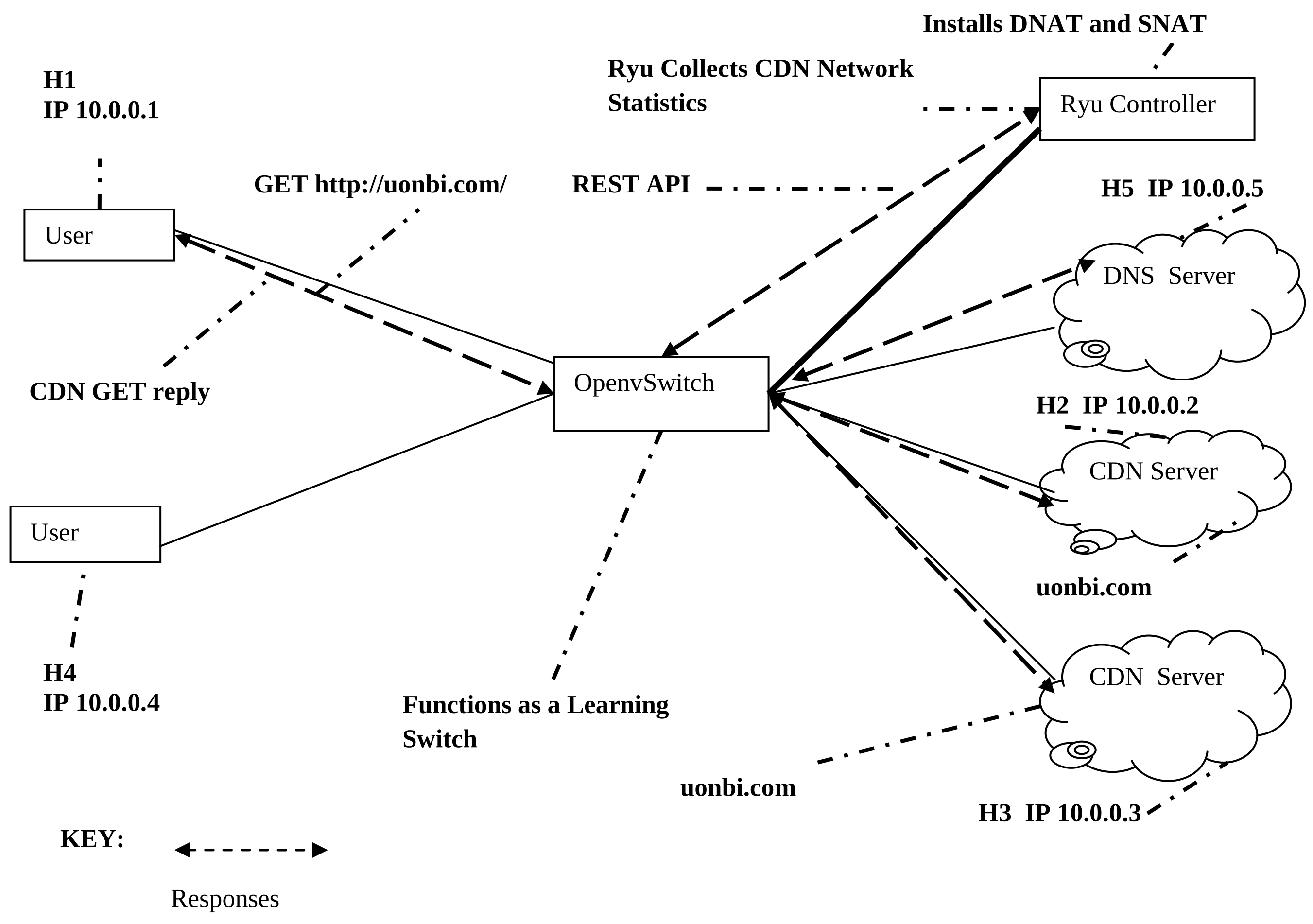


Fig. 2.2 Architectural Design

# CHAPTER THREE
# METHODOLOGY

## 3.1 Introduction

This chapter describes the overall network environment on which this project simulation tests were based and a complete approach as it apply to this study. The methods used are also described in detail. Section 3.2 explores that research design of this study. The experimental setup is found in section 3.3. Test procedures are shown in section 3.4.

## 3.2 Research Design

The study used descriptive design. A descriptive method entails observing and describing subject's behavior without exerting undue influence on them. In this study, an observation was done on twenty five tests carried out using Firefox and Chrome to determine user's access to uonbi.com content hosted in identical CDN replica servers. In the 25 tests carried out, the observation considered end user outbound requests which signaled content access at uonbi.com. The advantage descriptive design had over other research design in this study been diverse. It enabled researcher observer packets under investigations in a natural environment. This avoids issues of affecting their flow patterns. Further, descriptive research design simplified testing and measuring large samples required in qualitative experimentation. Also, it identifies variable constructs which might encourage further investigation by using other means. However, the limitation posed by this design is that no variables are manipulated; hence, a hindrance in analyzing results statistically. Further, observational results cannot be easily be repeated, a challenge in replicating experiment and reviewing results because setting is natural and all variables available cannot identify cause.

## 3.3 Experimental Setup

This study used quasi-experiment. This section describes the experimental environment used to design, implement and test DNS end user request redirection augmented with NAT.

### A. Topology Devices

Topology devices were emulated in a Mininet test bed. Ubuntu 14.04 acted as our operating system. It is important to note that the Ubuntu operating system does not operate an inbuilt DNS cache. The devices consisted of hosts. These hosts were labeled as H1, H2, H3, H4, H5, S1 and C0. H1 was designated to test our 'improved' CDN end user redirection mechanism. The host is within a CDN network. H1 was connected with 20Mbps to switch 1 to simulate an ADSL2+ high-speed subscriber line. H2 and H3 were our Web servers. They were designated as CDN replica servers. They served similar content. When not overloaded, they had 100Mbps connection to switch 1. When being simulated in a jammed environment, the connection speed was lowered down to 1Mbps. H4 was appointed as a host. It was used to contrast a standard CDN end user request redirection mechanism with H1 ('improved' mechanism). H4 was available outside a CDN network. This scenario was adopted to ensure H4 does not experience the effects of our proposed request redirection mechanism. H4 was connected with a 20Mbps connection to switch 1 to simulate an ADSL2+ high-speed subscriber line. H5 was used as our DNS server. S1 was our OpenvSwitch. We used C0 as our controller.

### a. C0 - Ryu Controller

Ryu was selected as our network controller framework for in this study. Ryu provides a rapid prototyping and testing platform for testing SDN OpenFlow controller applications. It has an inbuilt REST API. In this study, we used OpenFlow1.3 specification. The controller was used to communicate with the network switch. We implemented a REST API using a shell scripting language to push network changes to the controller. The Network Controller (C0) is connected to the CDN network to collect information. The information the controller collects includes CPU load, link load and network throughput. In our network, the controller pushes flow rules in the network switch when it boots. The flow rules allows the network controller feed the switch with network information collected to enable the switch redirects the packets to the correct port. This process is achieved by using a REST API. H1 and H4 are hosts representing end users in our network. They have 10.0.0.1 and 10.0.0.4 IP's respectively. Firefox and Google Chrome are used in both H1 and H4. For experimentation purposes, H1 is used for our request redirection. This allows us compare with H4 which uses DNS request redirection approach.

### C. S1 - OpenvSwitch

Our OpenvSwitch is an instance of OpenvSwitch running in a Mininet topology. OpenvSwitch is represented by S1 with an IP 10.0.0.6. We inspect flow tables any time using this switch by executing "ovs-ofctl –O OpenFlow13 dump-flows s1" command at command line (CLI). The switch is operated in OpenFlow1.3. Before the controller is started, the switch's flow table is empty, but when the controller is started, a flow table is inserted in the switch. OpenvSwitch is a learning switch with extra flow rules that allow NAT and DNAT insertion on TCP flows passing through it.

### D. H1 and H4 - Users

Our end users are represented using H1 and H4 in our network. They have 10.0.0.1 and 10.0.0.4 IP's respectively. We use H1 as a control to test our 'improved' CDN end user redirection mechanism while H4 is used to contrast the performance of our technique with H1. H1 has an IP address 10.0.0.1 while H4 has 10.0.0.4. H1 and H4 are both installed with Chrome and Mozilla Firefox. The browsers chosen are a representative of the proportion of the contemporary browser share market. For testing, we have our controller only apply the proposed request redirection mechanism to H1. A point to note is that the two browsers selected link pre-fetching and DNS caching of at least one minute.

### E. H5 - DNS server

This study was purposed to design and implement end user request redirection mechanism, and not the request redirection algorithm. Thus, the study designed a simple DNS server. The DNS was implemented on H5 using the BIND9. The DNS server was designated as an authoritative nameserver for uonbi.com content in our testing framework. When simulating network change (adding or changing optimal CDN replica server to the (Hostname), A record provided by H5 is updated using a shell script. In our experiment, this server has an IP address of 10.0.0.5

and sets the TTL on its response to be 30 seconds. Our CDN servers, H2 and H3 server identical content and point to a single directory (uonbi.com). It mirrors a CDN service provider having content spread across multiple servers.

**F. H2 and H3 – CDN Servers**

H2 and H3 represent our CDN server network. H2 and H3 are implemented using apache2 server files and renders access through port 80. They have IP addresses as 10.0.02 and 10.0.0.3 respectively. They both point to uonbi.com to render identical web pages.

**E. Links**

The links in our network topology had a configuration adding 10ms delay to TCP packets passing through them. The 10ms delay aimed at stimulating the location of all network components closer to the network edge router.

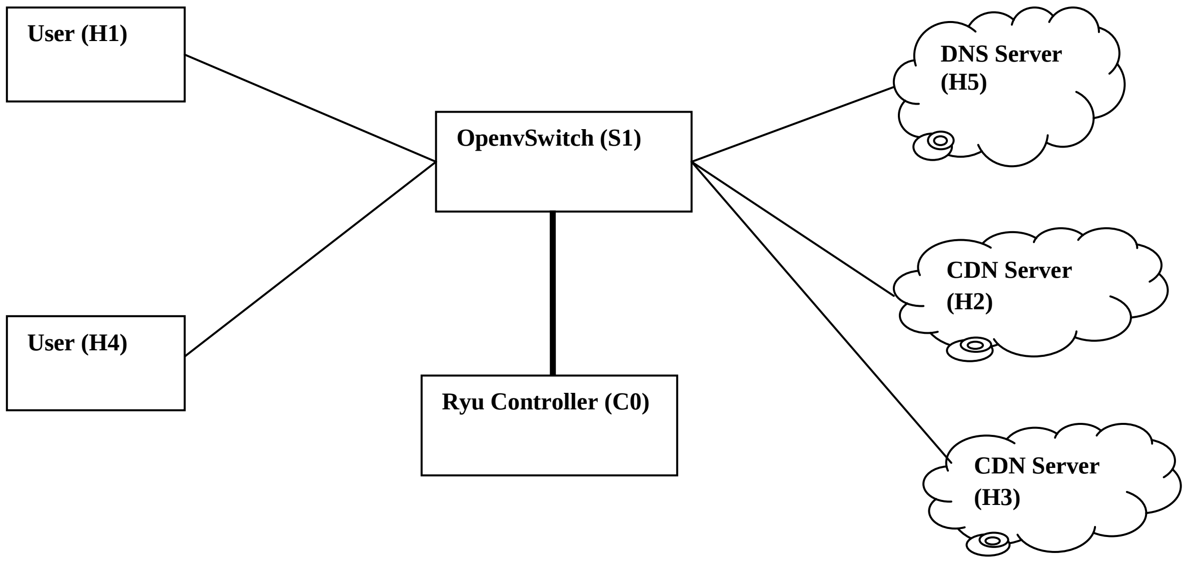


Fig. 2.3: Diagrammatic representation of network topology

**3.4 Main Method**

An end user wishing to access content hosted on a particular CDN, type a URL in a browser. The end user's browser then performs a DNS request, mapping an IP address to the URL typed in the browser. This process results in a request being serviced by the browser's DNS cache, with response returning instantly. The process prohibits external DNS servers from resolving the URL. The end user's browsers then connects to a newly established TCP session; communicating TCP packets with a CDN replica server. The newly established session is identified by source and destination TCP port, and the source and destination IP. When located within a local CDN network, for example, an ISP CDN network, the first packet reaches the edge router. However, if the destination IP address for this session is identical to the originating optimal server, mapped to a particular end user, then no further redirection will be required. Therefore, a route, establishing new flow for this TCP packet to a CDN server is established and the packet is redirected normally (see Appendix VI Wire shark Statistics). On the other hand, if the IP address for the TCP session differs from the IP of an optimal CDN server, end user redirection occurs. A flow is established, performing a DNAT that changes the destination IP packets belonging to this TCP session to a new CDN address. The TCP session then redirects the packets to an optimal CDN server, as a router normally does. A reverse route is the again established, performing SNAT on packets returning from a CDN server (see Appendix VI Wireshark Statistics).

| **Source** | **Destination** | **Protocol** | **Info** |
|---|---|---|---|
| 10.0.0.1 | 10.0.0.5 | DNS | Standardquery 0xc74c A uonbi.com |
| 10.0.0.5 | 10.0.0.1 | DNS | Standard query response 0xc17c A 10.0.0.3 |
| 10.0.0.1 | 10.0.0.3 | TCP | 53790 > http [SYN] Seq=0 |
| 10.0.0.3 | 10.0.0.1 | TCP | http>53790 [SYN, ACK] Seq=0 Ack=1 |

Fig. 3.1 A NAT process on a TCP packet flow

In Fig. 3.1, H1 with an IP address 10.0.0.1 make a request to 10.0.0.5, a DNS server for uonbi.com. A DNS provide response with IP 10.0.0.3. H1 (10.0.0.1) establishes a TCP-SYN to CDN server (IP 10.0.0.3). H1, 10.0.0.1, is assigned a TCP-SYN packet directed to 10.0.0.3 (CDN server) after DNAT has been attached. The packet trend continues until SNAT is attached to all the flow. The switch installs these flows, helping redirect end users based on the TCP session created. The TCP session established allows end users seamlessly connected, without being redirected midway (Appendix VI).

## 3.5 Test Procedures

This subsection explains the test performed in our experiment, the methods and explanation behind them. Thus, the test methods established were; request redirection after link failure, request redirection during link congestion and request redirection with the introduction of a new CDN replica server. Within each of these procedures, the study defined two types of tests. The tests were classified as a passive user and active user; with the ultimate goal of attaining end user performance in a CDN network. The tests were desirable so as to model 'expected' user behavior in times of duress. As such, the study modeled a scenario where users access content by typing the URL into their browser and another scenario where if congestion was experienced, or load time was greater than 20 seconds, the user refreshes the page.

**A. Request redirection after link failure**

In this method, the study observed the effectiveness of our proposed request redirection mechanism after a CDN replica server encountered a link failure. The primary concern in this test was content access delay.

**B. Request redirection when the link is congested**

In this test, the study observed the success of the proposed request redirection approach when a CDN replica server was under congestion.

**C. Request redirection after introducing a new CDN replica server**

This test aimed at providing information on the speed at which a new CDN replica is established depending on server request and how it affects visitors on a CDN network.

## 3.6 Software's and Tools

### 3. 6.1 System Requirement

The approach was modeled on a Virtual Machine with a base memory of 1024MB with a hard disk space of 100.70 GB

### 3. 6.2 Virtual Box

Oracle VM Virtual Box 5.1.3 was used. The VM is where our test environment was set.

### 3.6.3 Ubuntu Operating System

Our research used ubuntu-14.04.4 as a test platform. The platform enabled the researcher install other important components such as Mininet.

### 3.6.4 Apache2

The research used Apache as an HTTP server because of its popularity. Apache renders webpages requested by web browsers

### 3.6.5 Bind9

Bind9 was used to provide caching to subsequent DNS queries. The DNS server remembered all the domain queried and enabled further request to be rendered locally.

## 3.7 Sampling

The target population considered by this study was packets (GET, TCP, DNS and HTTP) from end users (H1 and H4). The unit adopted for analysis was individual packets. Performance tool was used to randomly generate twenty five test samples. Since "sniffing" of end user requests begun at H1 and H4, the objectives of collecting test sample at this point were to make it efficient and secure. This means that only valid tests on end user requests send out towards CDN replica servers for starting/accessing uonbi.com contents was collected. Tests on the data gathered were done at intervals. The test samples generated were used as the sampling frame. The packet sampled provided a reasonably representation of users in a CDN network. Random sampling was appropriate owing to the uniqueness of this study.

## 3.8 Data Collection

### 3.8.1 Observation

A systematic observation of GET, TCP, DNS and HTTP packet behaviors during the experiment was recorded at intervals. Since "sniffing" of end user requests begins at H1 and H4, the objectives of collecting packets at this point was to make it efficient and secure. This meant that only valid tests on end user requests send out towards CDN replica servers for starting/accessing uonbi.com contents were collected. Observation on tests regarding data gathered was done at intervals. HTTP GET requests were only collected using a web performance tool available in

Mozilla Firefox and Chrome. Observation method was highly favored in this study. It required no demand characteristics or subject reactivity.

### 3.8.2 Data Sources

The study utilized primary data sources. Primary data was obtained by observing GET, TCP, DNS and HTTP packets from the experiment carried in a Virtual Machine (VM). Critical data originated from five principal sources. The first source was the end users. The end users were the client hosts. They were labeled as H1 (10.0.0.1) and H4 (10.0.0.4). Data from H4 consisted of DNS request packets directed towards a CDN replica server and subsequent DNS response from the same CDN replica server. The second data source was the DNS servers. The DNS servers generated TTL responses to H1 and H4 requests. They also stored content accessed by the end users. The third data source was the CDN replica servers. Since this study focused on request redirection technique the CDN replica servers stored identical content on multiple servers (unonbi.com). The data generated included responses taken to access a given type of content, for instance, dynamic or static data. The fourth data source was the SDN switch. SDN switch was used to provide flow table data based on network information. The final data source was the links. Links were used to provide data on link congestion.

## 3.9 Data collection tools

This section describes the tools a researcher used to gather data

### 3.9. 1 Wireshark

The experiment used Wireshark as a tool to capture and analyze network statistics from our experiment.

### 3.9.2 Web performance tool

Mozilla Fireworks and Chrome contain a performance tool for viewing web performance in real time. The tool is used to view HTTP requests from the source, network performance and other messages. The performance tool contains sub tools such as such as the Call Tree to help analyze the browser’s HTTP request performance. All the processes are achieving using a recording feature.

## 3.10 Data Quality

Reliability and validity are key criteria used in determining data integrity. In this study, the researcher used reliability and validity to ensure data collected was valid.

### 3.10.1 Reliability

To maintain the reliability of the study, the researcher administered 25 tests on all end users (H1 and H4) running Mozilla Firefox and Chrome browsers. The assumption was that both hosts have similar conditions during the test. Twenty five samples, randomly selected to guarantee external validity. Further, an elaborate recording criteria using performance tool was administered to increase reliability.

### 3.10.2 Validity

The validity of an instrument is determined when it measures what it is perceived to measure accurately.  To achieve validity, the researcher analyzed literature in relation to valid instruments to measure website performance. This was to get more insight on the appropriate tool to guarantee research validity. The instruments the researcher used were also pre-tested using a sample data to ensure they align to research objectives.

## 3.11 Data Analysis Methods

This study used descriptive statistics to perform analysis of gathered data

### 3.11.1 Data analysis and presentation

The research was interested in tracking “GET” request emanating from H1 and H4 to assess end user request redirection performance in CDNs. Analysis entailed examining and interpreting, placing collected within its structure and constituent elements.  The key data for the researcher was the GET, DNS and HTTP data. These data were important in assessing the performance of H1 and H4 end user request redirection technique using our new approach.  Also, these data was helpful in assessing Mozilla Firefox and Chrome performance in terms of caching end user requests.  As the study used descriptive design, data gathered and the behavior of events were organized, tabulated and represented in form of tables and line charts.  Tables and line graphs charts were chosen to assist readers understand distribution of data.

# CHAPTER FOUR
# RESULTS AND DISCUSSION

## 4.1 Results

This chapter presents performance results obtained from our end user request redirection. Our performance results were based on three measurements. The first measurement was end user request redirection after a link failure, end user request redirection during link congestion and end user request redirection with the introduction of a new CDN server.

### 4.1.1 End User Request Redirection after Link Failure

The test was executed to assess our proposed approach effectiveness when a link to CDN server fails. Emphasis was placed on content access delay. The results were determined using Firefox and Chrome. The first parameter employed was the reference time required to load the website located in the CDN replica server without invoking request redirection. When Google Chrome and Firefox browsers were used passively as a testing approach, they were unable to fully load a page after one to two minutes. They were therefore categorized as "timeouts". However, when active invocation was done, Firefox and Google Chrome showed a significant outcome on the load time. Firefox recorded a mean of 80.00 seconds difference in the load time compared to Chrome. A similar experiment with Firefox also achieved 120.10 seconds, thus creating a difference of 40.1 seconds. For instance, after 25 seconds of inactivity, a refresh was performed. When results were not forthcoming, a second refresh was executed at 30.05 seconds. This was done continuously until the uonbi.com began loading.

The Chrome average mean load was 30 seconds compared to that shown by Firefox. The active behavior was similar. The refresh rate was 30, 35 and 150.05 based on the simulation observation. In this case, the access delay for the page was a minute after a refresh was done after the total of about a minute's delay since the browser visited uonbi.com, the first test page appeared to begin loading. However, despite the starting to load, it did not load making three images out of six to load on the academic.html. The page was refreshed after 30 seconds and loaded successfully. However, the academic.html did not load successfully. This behavior was consistent and was a major contributor to extra time in subsequent observation as the table below indicates.

Table 4.1 Table illustrating the time in seconds (s) browsers took to traverse and load uonbi.com website

| **Variable** | **Firefox** | | **Chrome** | |
|---|---|---|---|---|
| | **Mean** | **Standard Deviation** | **Mean** | **Standard Deviation** |
| Mean Load Time | 25.0 | 0.08 | 30.0 | 0.08 |
| Improved Request Redirection Technique | 30.05 | 4.2 | 29.50 | 3.4 |
| Request Redirection Technique ( Passive User) | - | - | - | - |
| Improved Request Redirection Technique – Active User | 80.00 and 120.10 | 0.6 | 150.05 | 4.196 |

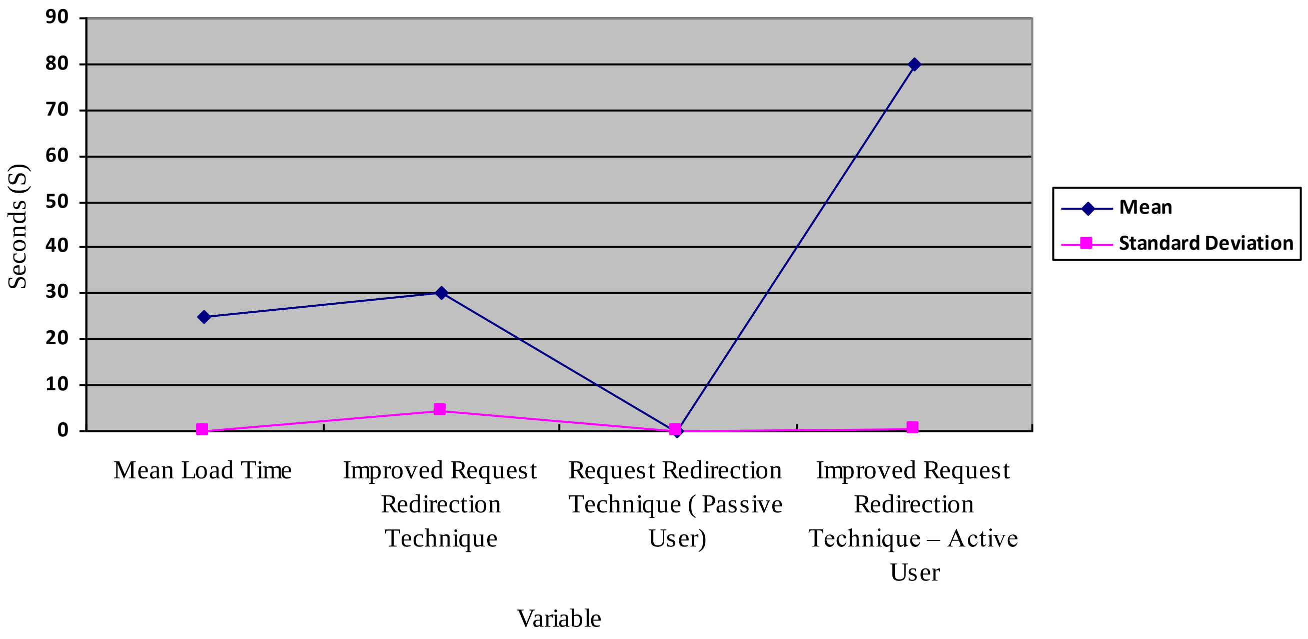


Figure 4.1 Time in seconds (s) Firefox browser took to traverse and load uonbi.com website

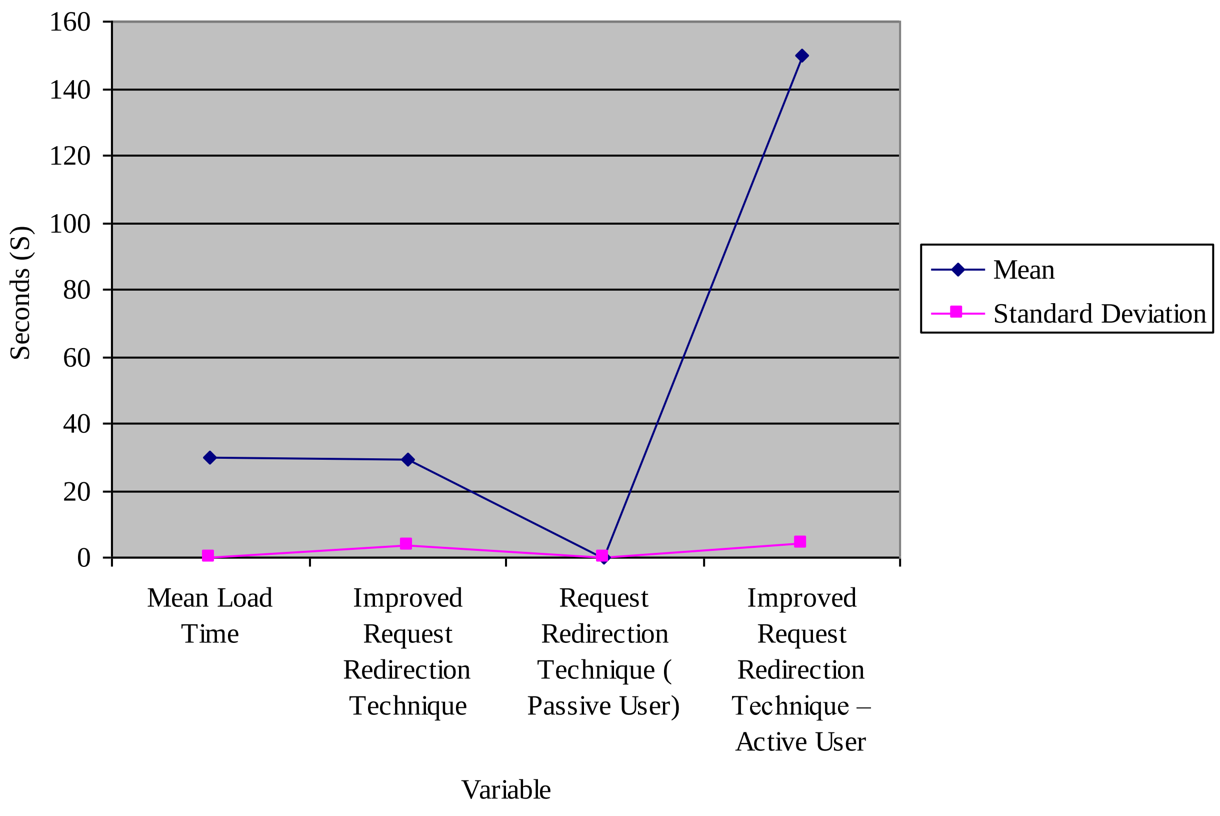


Figure 4.2 Time in seconds (s) Chrome took to traverse and load uonbi.com website

**4.1.2 End User Request Redirection during link congestion**

The results for request redirection during link congestion were carried out to assess our approach when a CDN server was experiencing heavy congestion. The tests were simulated using two methods. These methods were when the link was uncongested (similar to 4.1.1) and when the link was congested. Mean load time when a link was congested designated the duration for uonbi.com to load all pages on the congested link with 0 end user request redirection. In a test to assess load time without improved request redirection mechanism, Chrome recorded a mean of 86.51 seconds and Firefox recorded 84.23 seconds. However, both browsers recorded a delay when connecting to a new CDN server without using improved request redirection. Firefox recorded the highest mean of 71.20 while Chrome had 59.04. However, the time taken to connect to a new CDN server using an improved request redirection technique was low with Firefox recording a mean of 15.71 seconds while chrome showed 10.05 seconds. Further, load time using an improved request redirection differed among the browsers. Firefox recorded a mean of 29.05 while Chrome recorded 30.40 seconds.

Table 4.2 Duration in seconds a user connects to an optimal CDN replica server and the time taken to traverse uonbi.com website.

| **Variable** | **Firefox** | | **Chrome** | |
|---|---|---|---|---|
| | **Mean** | **Standard Deviation** | **Mean** | **Standard Deviation** |
| Mean Load Time | 26.9 | 0.02 | 26.70 | 0.07 |
| Mean Load Time – Congested | 120.6 | 0.03 | 125.08 | 0.09 |
| Load Time – Improved Request Redirection Technique | 29.05 | 0.9 | 30.40 | 0.19 |
| Time until connected to new CDN server, with improved request rerouting mechanism | 15.71 | 0.06 | 10.05 | 0.16 |
| Load time without improved request redirection mechanism | 84.23 | 0.01 | 86.51 | 2.42 |
| Time until connected to new CDN server, without improved request redirection mechanism | 71.20` | 0.06 | 59.04 | 0.07 |

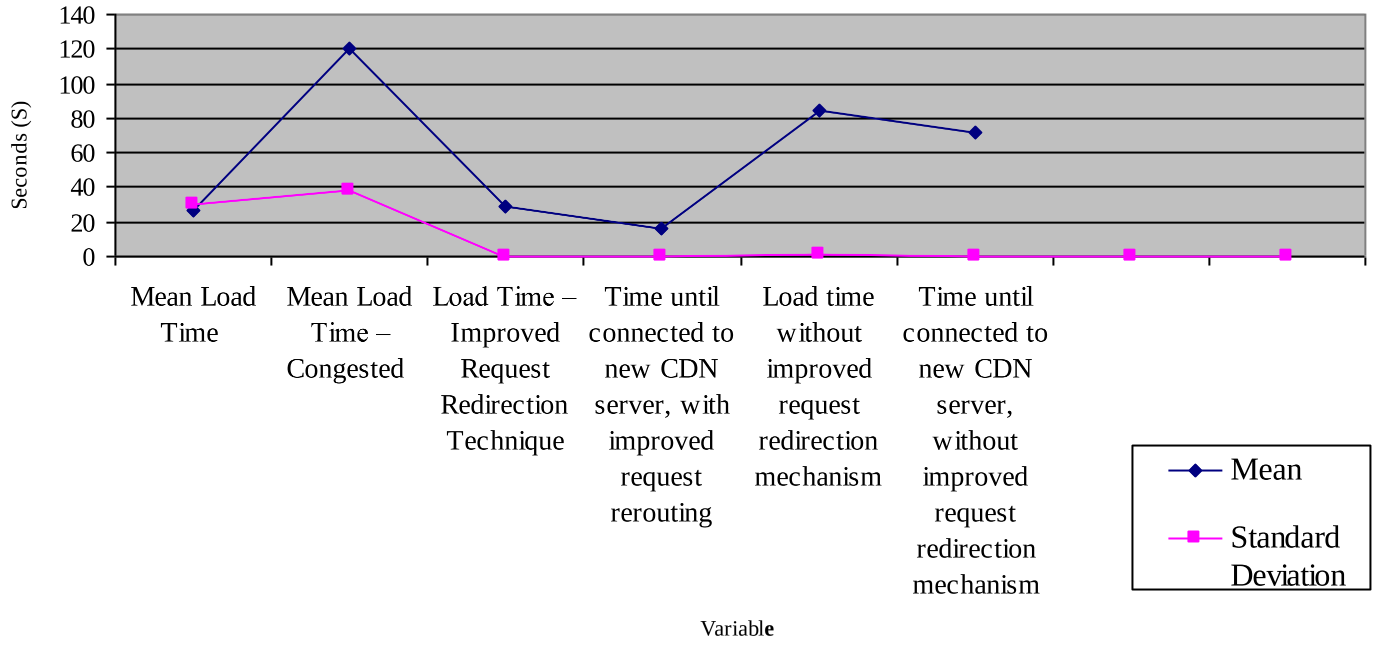


Fig 4.3 Time in seconds a user connects to an optimal CDN replica server using Firefox and the time taken to traverse uonbi.com website.

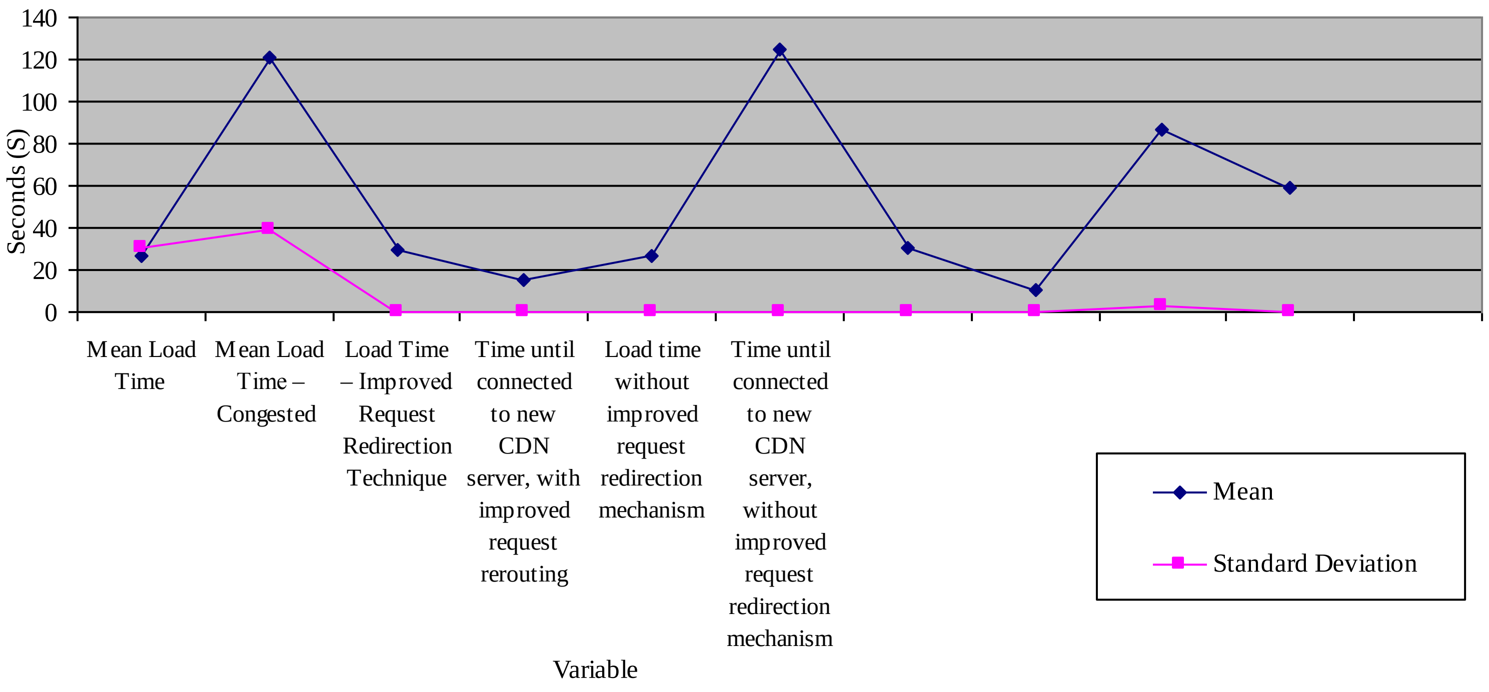


Fig 4.4 Time in seconds a user connects to an optimal CDN replica server using Chrome and the time taken to traverse uonbi.com website.

### 4.1.3 End User Request Redirection with the introduction of a new CDN server

The load reference time in request redirection with the introduction of a new CDN server determined the duration H1 and H4 hosts took to complete loading uonbi.com website. This is when the hosts were accessing a congested H2 CDN replica server simultaneously. When the measurement was taken, H1 and H4 appeared to load the uonbi.com randomly besides taking turns downloading H2 content. Request redirection with the introduction of a new CDN server measured H2 total time when it was subjected under the load. Further, even if a single host had completed uonbi.com test pages, it was noted that the time was lower than a half achieved compared to h4.

The higher standard deviation values observed in other measurements are as a result of H1. If H1 was flooded with GET request responses emanating from H2 at the 40-second mark, it could take 15 to 20 seconds for H1 to complete the uonbi.com. As the Table shows, the median time H1 takes to complete uonbi.com test pages is half or less compared to that of H4. Moreover, the mean time exhibited by H4 to complete loading uonbi.com websites is a half of the reference time.

Table 4.3 Duration in seconds H1 and H4 take to load uonbi.com website

| | **H1** | **H4** | |
|---|---|---|---|
| Mean Load Time – (H1 & H4) | Mean 148: Std 1.16 | | |
| Average mean load time (H1 & H4) | Mean 60.9: Std 19.4 | Mean 140.2: Std 18.7 | |

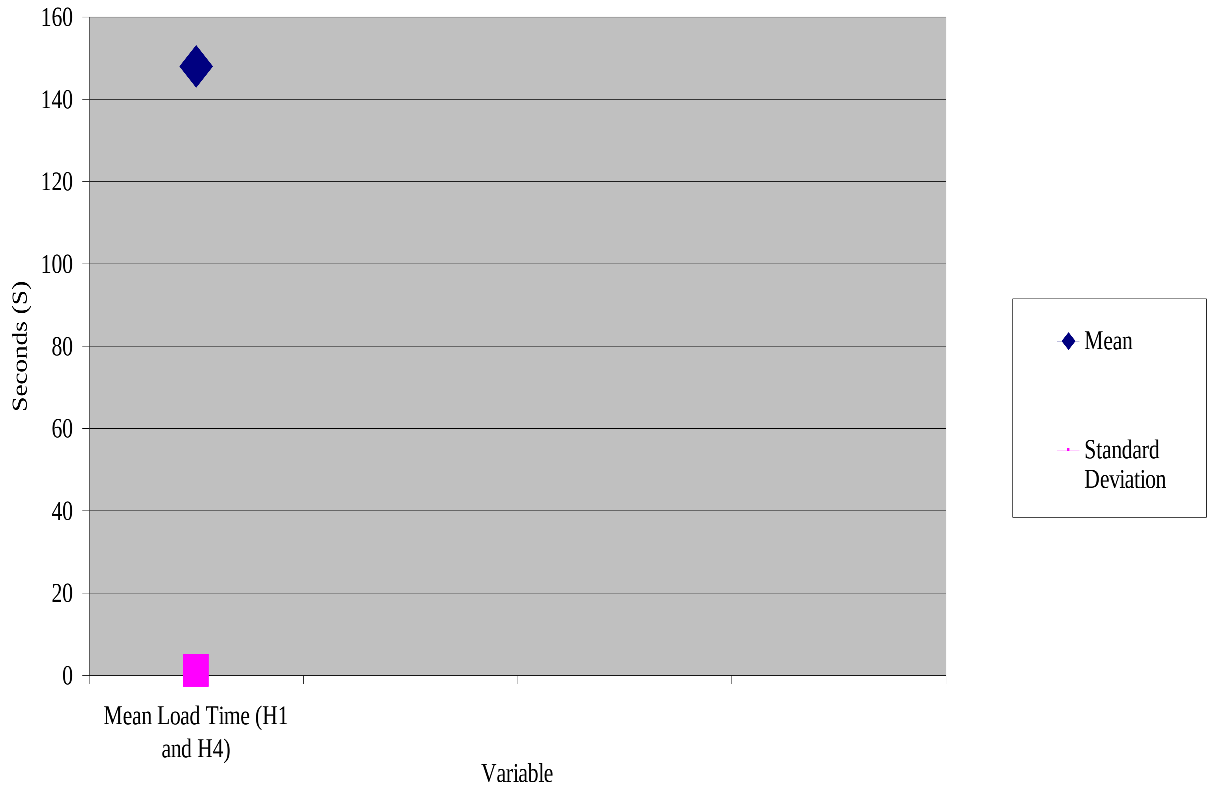


Fig 4.5 Duration in seconds H1 and H4 take to load uonbi.com website

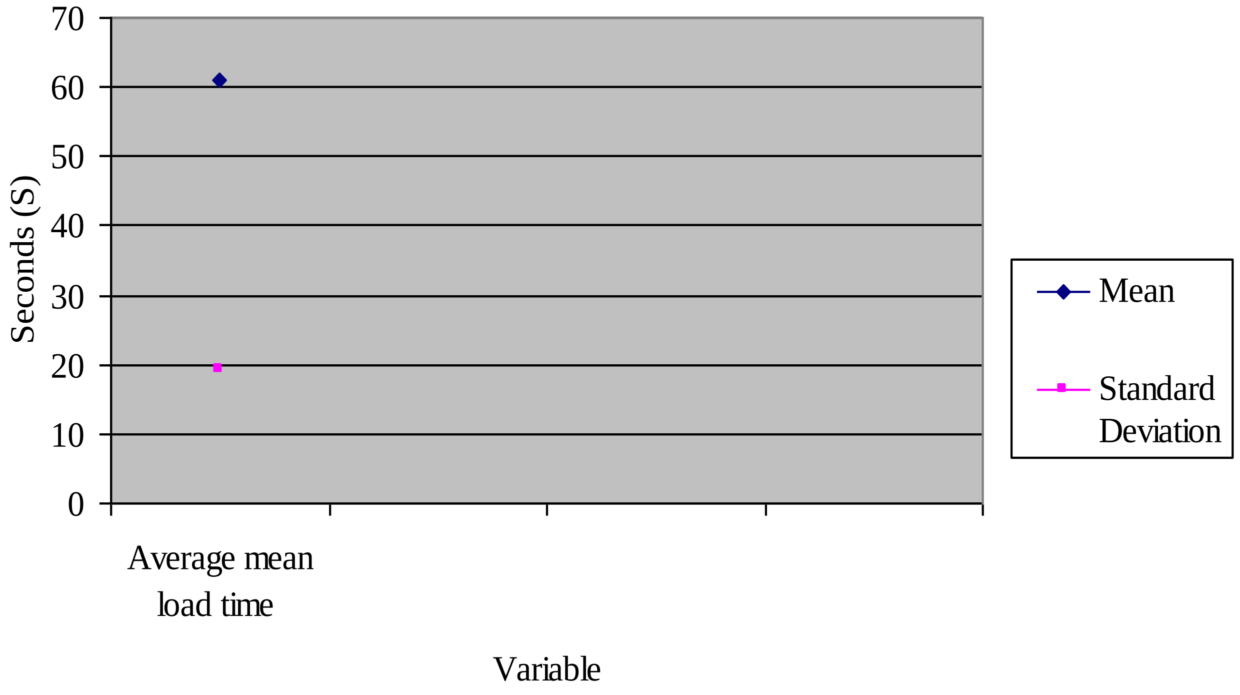


Figure 4.6 Duration in seconds H1 take to load uonbi.com website

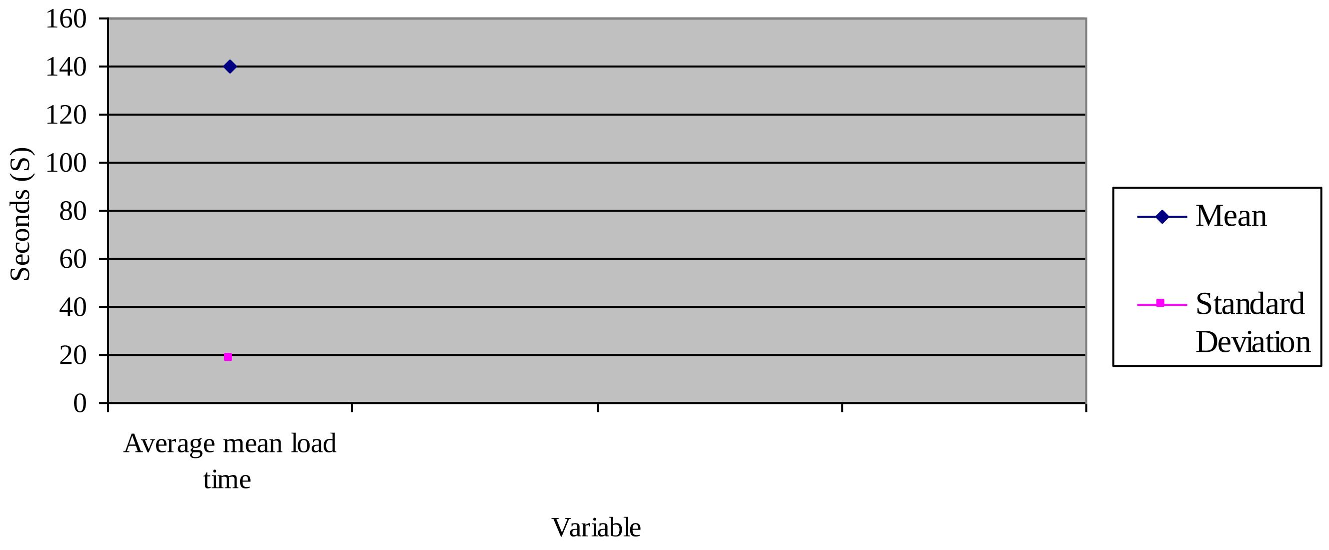


Fig 4.7 Duration in seconds H4 take to load uonbi.com website

## 4.2 Discussion

From the results, different patterns of behavior were observed in Firefox and Chrome browsers when active test mode was executed. Firefox recorded a mean load time of 80.00 seconds and 120.00 seconds respectively on the results gathered. The cause of this behavior was linked to page refreshes after a period of inactivity. Subsequent refreshes on uonbi.com at some intervals made unonbi.com to load (Table 4.1) these refreshes were consistent with the 80.00 seconds and 120.00 seconds mentioned in Table 4.1. As demonstrated in table 4.1, the mean load time for Chrome was 30.0 seconds while Mozilla Firefox was 25.0 seconds. Further, the active behavior was also evident when refresh was executed at 20 seconds until an activity was observed. This action resulted in access delay in a page lasting for a minute when the page was refreshed since after the browser visited uonbi.com, this activity resulted in uonbi.com homepage to begin loading. The behavior described appeared to be consistent contributing to extra time shown in table 4.1.

The results demonstrate that our proposed end user request redirection technique is able to redirect end users successfully after a link has failed. On the other hand, there is an observable performance cost as indicated by the mean load time. The mean load time is 5.05 seconds higher to our proposed end user redirection technique compared to the reference.

The results investigating end user request redirection when the link is congested demonstrates that both Mozilla Firefox and Chrome achieved a close mean load time of 26.9 and 26.70 seconds respectively (Table 4.2). This confirms they both navigated the congested reference mean load time. This is evidenced by changing connections from congested H2 to H3 which was uncongested between academic.html and admissions.html page. In the table Table 4.2 Mozilla Firefox and Chrome shows a significant discrepancy (71.20 and 59.04 seconds) delay until when an end user connects to uncongested H3 server. Firefox only connected to H3 between the page loads while Chrome connected to H3 by loading academic.html half-way. Our results demonstrate that our end user request redirection technique is able to provide end users with successful redirection by making use of current network information. However, the mean load time is higher than the reference time (0.2 seconds). This observation indicates that end users that are successfully redirected by our approach have their initial TCP packets traverse through the controller. This ensures they access optimal CDN replica servers. The processing of TCP packets by the controller might be the reason causing the 0.2 seconds lag.

The results for request redirection with the introduction of a new CDN server demonstrate higher standard deviation values unlike other previous measurements. The higher values could be linked to H1. When H1 is flooded with GET request responses emanating from H2, it could take more time for H1 to complete the uonbi.com. Similarly, the median time H1 takes to complete uonbi.com test pages is less compared to that of H4. Moreover, the mean time exhibited by H4 to complete loading uonbi.com testpages is a half of the reference time. This demonstrates that our proposed end user request redirection can successfully be tailored to use the introduction of rapidly established CDN replica servers to augment Quality of Experience for users and CDN network by indirectly reducing link overloading.

Our proposed end user request redirection was a success. Content load time was reduced compared to traditional DNS approach. Results achieved by investigating side effects of client side DNS caching are perplexing. Concerns such as the timing effects of a refreshed page and how quickly that page refreshes is still a paradox to the researcher. While the author expected errors such as page timeouts as a result of DNS caching, timing errors were not. Such unexpected behavior is why this research was undertaken.

Majority of previous studies have dwelt on distributed redirection schemes where end users access content from a CDN replica server by performing redirection closer to client machines. These studies found out that this scheme reduced improved end user request redirection by minimizing server loads and network overheads. However, the shortcomings of these studies were insufficient evaluation of their approaches. This study has taken the approach of DNS augmented with NAT technique and confirms that end user request redirection can be improved when client side DNS caching is kept at minimal. This has the potential of reducing server loads as well as network overheads.
Other studies have used a modified DNS scheme to improve end user request redirection in CDNs. The DNS modified scheme pushed end users request redirection closer to clients using locally available DNS servers. Their results confirmed that a modified DNS scheme was successful in returning possible list of servers together with their valid load information. This study strengthens the effects caused by DNS caching. DNS augmented with NAT technique has the capability of bypassing DNS caching improving end user redirection.

# CHAPTER FIVE

# CONCLUSION

## 5.0 Conclusion

In this study, we investigated end user redirection performance in Content Development Network (CDN) using Software Defined Networking (SDN)-based network. The study has successfully demonstrated SDN based approach can be used to redirect users away from accessing content on a non-optimal CDN server to a better performing CDN replica server. This might be the case when the network changes or if the user side contains stale DNS information as a result of client side DNS caching. The study used OpenFlow switch running within a virtualized Mininet network. A network controller was used to enable end user request redirection by installing Network Address Translation (NAT) and Destination Network Address Translation (DNAT) flow rules that directed TCP packets.

**I. To investigate effects of client-side DNS caching on the contemporary DNS-based redirection technique**

This study sought to investigate the effects of client-side DNS caching on contemporary DNS-based end user redirection. The Table 4.1 shows that the Mozilla Firefox and Chrome when subjected to passive testing, they failed to load a uonbi.com page hence, being timeout. Further, (4.1.1) shows that end users had to refresh their browsers for over a minute to get a response from a CDN replica server. Similarly, after 25 seconds of inactivity, a refresh was performed on both browsers. When results were not forthcoming, a second refresh was executed at 30.05 seconds. This was done continuously until the uonbi.com began loading. These observations confirm our claim regarding the client-side effects on end user redirection; hence, this study has successfully achieved this objective.

**II. To investigate SDN based technique in enhancing quality of service in end user request redirection**

In 4.1.3 showed that when simultaneous end users (H1 and H4) are accessing CDN replica servers, it could take the least time (ten seconds) to download contents from CDN replica server H2 and get redirected. When H1 is redirected, it leaves H4 to use the “now” available uncongested link (4.1.2). Further, when connecting to a new CDN server using our improved approach, Firefox took a mean time of 15.71 seconds while Chrome 10.05 seconds. This therefore, demonstrates that our proposed technique improves performance by increasing Quality of Experience of end users as well as reducing loads on the CDN network. From the study, this goal has been achieved.

**III. To design a user request redirection method using SDN that can redirect users reliably when it recognizes DNS contains out of date information.**

The study designed an end user request redirection using SDN based technique. The design used the Domain Name System augmented (DNS) with Network Address Translation (NAT) Technique. In the design, a network controller gathered the network information and installed SNAT and DNAT flow rules that directed TCP packets. This technique was able to reliably redirect users to an optimal CDN server. For example, when a link fails, the technique is able to redirect users to a reliable and optimal server.

**IV. To evaluate the effectiveness of the proposed user request redirection technique in improving end user request redirection performance.**

The proposed technique aimed at: working around the effects of client side DNS caching, simplifying networking configuration and making use of network information. The results demonstrated in Table 4.1 assert the effects caused by client side caching. This effect is illustrated by "timeouts" and frequent page refresh. Further, our approach demonstrates that our proposed end user request redirection technique is able to navigate client site caching redirect end users successfully (Table 4.1). Also, our approach used network controller, incorporated shell scripts and REST APIs. These techniques confirm that our proposed design is easier to configure and integrate with the network, hence fulfilling this goal. Finally, on making use of network information, our end user request redirection fulfills this goal.

## 5.1 Future Work

This study focused majorly on improving end user access to content stored in a CDN, a logical continuation of this work will be designing and implementing end user redirection algorithm that can make use of end user redirection approach proposed by this study. Further, live implementation of the proposed technique to ascertain its scalability provides another area of improvements. This is because the study used a virtualized environment to carry out end user request redirection performance tests.

## References

Acker, S. Van, Hausknecht, D. & Sabelfeld, A., 2016. Data Exfiltration in the Face of CSP. *ACM*, 4(6), pp.853–864.

Aggarwal, V., Feldmann, A. & Scheideler, C., 2007. Can ISPS and P2P users cooperate for improved performance? *ACM SIGCOMM Computer Communication Review*, 37(3), p.29.

Akamai, 2016. *Request Router A software-based routing and redirection service within a comprehensive CDN solution.* Technical Report

Akyildiz, I.F. et al., 2014. A roadmap for traffic engineering in software defined networks. *Computer Networks*, 71, pp.1–30.

Baggio, A., 2004. *Distributed redirection for the Globule platform*. IR-CS-010

Baggio, A., 1999. *Yet another redirection mechanism for the World-Wide Web ?*

Calder, M. et al., 2015. Analyzing the Performance of an Anycast CDN. *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pp.531–537.

Calin, D. & Schulzrinne, H., 2015. Intelligent content delivery over wireless via SDN. *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, pp.2185–2190.

Calvin, M., Arun, D.R. & Balamurugan, P., 2016. Balancing Work Load of a Cloud and Dynamic Request Redirection for Cloud Based Video Services Using CDN and Data Centre. , pp.1–6.

Chen, F., Sitaraman, R.K. & Torres, M., 2015. End-User Mapping: Next Generation Request Routing for Content Delivery. *Acm Sigcomm*, 4(9), pp.167–181.

Chen, J. et al., 2016. Forwarding-Loop Attacks in Content Delivery Networks . *Ndss*.

Cicalese, D. et al., 2015. A First Look at Anycast CDN Traffic. *Acm Sigcomm*, 3(September), pp.1–9.

Cisco, 2016. *Cisco Visual Networking Index (VNI) Update*, Cisco Visual Networking Index: Forecast and Methodology, 2015–2020

Clara, M.S. et al., 2016. NSDI ' 16 : 13th USENIX Symposium on Networked Systems Design and Implementation. In *13th USENIX Symposium on Networked Systems Design and Implementation*.

Elkotob, M. & Andersson, K., 2012. Challenges and opportunities in content distribution networks: A case study. In *GC'12 Workshop: The 4th IEEE International Workshop on Mobility Management in the Networks of the Future World Challenges*. pp. 1021–1026.

Fei, H., 2014. Network Innovation through OpenFlow and SDN: Principles and Design. , p.520. Available at: http://books.google.com/books?id=f5WlAgAAQBAJ&pgis=1.

Flavel, A. et al., 2015. FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs. *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 27, pp.381–394.

Frank, B. et al., 2012. Content-aware Traffic Engineering. Available at: http://arxiv.org/abs/1202.1464.

Frank, B. et al., 2013. Pushing CDN-ISP collaboration to the limit. *ACM SIGCOMM Computer Communication Review*, 43(3), p.34.

Garg, A., 2015. Analysis of various techniques for improving Web performance. *Ijarcsms*, 3(3), pp.271–279.

Georgopoulos, P. et al., 2014. Cache as a service: Leveraging SDN to efficiently and transparently support video-on-demand on the last mile. In *Proceedings - International Conference on Computer Communications and Networks, ICCCN*. pp. 1–9.

Ghasemi, M. et al., 2016. Performance Characterization of a Commercial Video Streaming Service. *IEEE Internet*

*Computing*, 16(5), pp.1–15.

Giancarlo Fortino, C.E.P., 2012. Next Generation Content Delivery Infrastructures : Emerging Paradigms and Technologies. *IGI Global*, 34(16), p.316.

Gupta, M. & Kumar, D., 2014. State-of-the-art of Content Delivery Network. *International Journal of Computer Science and Information Technologies*, 5(4), pp.5441–5446.

Handigol, N. et al., 2010. Aster * x : Load-Balancing Web Traffic over Wide-Area Networks. *Deutsche Telekom R&D*, 12(8), p.3.

Harahap, E. & Wijekoon, J., 2013. Modeling of Router-based Request Redirection for Content Distribution Network. *International Journal of Computer Applications*, 76(13), pp.37–46.

Hutchison, D., 2014. Passive and Active Measurements. In *15th International Conference, PAM 2014 Los Angeles, CA, USA, March 10-11, 2014 Proceedings*. p. 124.

Juneja, D. & Garg, A., 2012. Collective Intelligence based Framework for Load Balancing of Web Servers. , 3(1), pp.64–70.

Kende, M., 2014. Internet Society Global Internet Report 2014. *Internet Society*, p.146. Available at: http://www.internetsociety.org/doc/global-internet-report.

Liang, P.-H. & Yang, J.-M., 2015. Evaluation of Two-Level Global Load Balancing Framework in Cloud Environment. *International Journal of Computer Science and Information Technology*, 7(2), pp.1–11.

Liu, C. et al., 2012. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network. *Signal Processing: Image Communication*, 27(4), pp.288–311.

Mein, P.A., 2012. *A Latency-Determining / User Directed Firefox Browser Extension*. University of Kansas.

Nygren, E., Sitaraman, R.K. & Sun, J., 2010. The Akamai network: a platform for high-performance internet applications. *SIGOPS Operating Systems Review*, 44(3), pp.2–19.

Olteanu, V. et al., 2015. In-Net : In-Network Processing for the Masses. *EuroSys*, pp.1–15.

Ooka, A. et al., 2013. Networking Architecture and Router. In *Future Network and MobileSummit 2013 Conference Proceedings*. pp. 1–10.

Puopolo, S. et al., 2011. *Content Delivery Network ( CDN ) Federations - How SPs Can Win the Battle for Content-Hungry Consumers*,

Ranjan, S. & Knightly, E., 2008. High-performance resource allocation and request redirection algorithms for web clusters. *IEEE Transactions on Parallel and Distributed Systems*, 19(9), pp.1186–1200.

Safavi, M. & Bastani, S., 2015. A Simulation Package for Energy Consumption of Content Delivery Networks (CDNs). In *Proceedings of the "OMNeT++ COmmunity Summit 2015*. pp. 4–6.

Sahri, N.M. & Okamura, K., 2014. Openflow Path Fast Failover Fast Convergence Mechanism. , 38, pp.23–28. Available at: http://dx.doi.org/10.7125/APAN.38.4.

Saini, R., 2015. A Hybrid Algorithm for Load Balancing. *International Journal of Advanced Research in Computer Science and Software Engineering Research*, 5(7), pp.3–8.

Schmidt, R.D.O., Heidemann, J. & Kuipers, J.H., 2016. Anycast Latency : How Many Sites Are Enough? *ISI-TR*, (May).

Shaikh, a., Tewari, R. & Agrawal, M., 2001. On the effectiveness of DNS-based server selection. *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, 3, pp.1801–1810.

Silvestre, G. et al., 2013. Predicting popularity and adapting replication of internet videos for high-quality delivery. *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, pp.412–419.

Triukose, S., Wen, Z. & Rabinovich, M., 2011. Measuring a Commercial Content Delivery Network. *Proceedings of the 20th international …*, pp.467–476. Available at: http://dl.acm.org/citation.cfm?id=1963472.

Trossen, D. et al., 2016. *European Technology Platform for Communications Networks and Services NetWorld2020 ETP Expert Working Group on*, Available at: networld2020.eu/wp.../NetWorld2020_Joint-Whitepaper-V8_public-consultation.pdf.

Vasilakos, A. V. et al., 2015. Information centric network: Research challenges and opportunities. *Journal of Network and Computer Applications*, 52, pp.1–10.

Wang, R., Butnariu, D. & Rexford, J., 2011. OpenFlow-Based Server Load Balancing Gone Wild Into the Wild : Core Ideas. *Hot-ICE'11 Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and servicesworks and services*, p.12.

Weaver, N. et al., 2011. Implications of Netalyzr ' s DNS Measurements. *Icsi.Berkeley.Edu*, (II), pp.1–8.

Whitaker, A., Shaw, M. & Gribble, S.D., 2002. 5th Symposium on Operating Systems Design and Implementation. In *The Effectiveness of Request Redirection on CDN Robustness LiminWang,*. p. 33.

Wytrębowicz, J. et al., 2014. SDN Controller Mechanisms for Flexible and Customized Networking. *International Journal of Electronics and Telecommunications*, 60(4), pp.299–307.

Xiang, C., Junyong, T. & Yong, Z., 2015. Towards a Content Delivery Load Balance Algorithm Based on Probability Matching in Cloud Storage. , pp.2211–2217.

Xu, C., Chen, B. & Qian, H., 2015. Quality of Service Guaranteed Resource Management Dynamically in Software Defined Network. *Journal of Communications*, 10(11), pp.843–850.

Yu, M. et al., 2010. Scalable Flow-Based Networking with DIFANE. *SIGCOMM*, 6(3), pp.4–10.

## 6.1 Appendix

### I. Rest API Controller Sample Code

```
#!/usr/bin/env python
import json
import logging
from operator import attrgetter
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER,DEAD_DISPATCHER, CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_els
from ryu.lib import hub
from ryu.lib import dipid_lib
from webob import Response
from ryu.app.wsgi import ControlerBase, WSGIApplication, route
from ryu.lib.packet import ethernet, tcp
from ryu.lib.packet import packet

simple_switch_instance_name = 'simple_switch_api_app'
url = '/simpleswitch'/add/{dpid}
url1 = '/simpleswitch/remove/{dpid}'
flag = 0

class cdnController(app_manager.RyuApp):
        _CONTEXTS = {'wsgi': WSGIApplication}
        def __init__(self) :
         super(cdnController, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.mac_to_port = {}
        wsgi = kwargs ['wsgi']
        wsgi.register(SimpleSwitchController, {simple_switch_instance_name : self})
        self.switches = {}
        self.flows = {}
        self.logger.info ("Loading cdnController")

        #Initializer adds the table-miss flow entry
        uset_ev_els (ofp_event.EventOFPSwitchFeatures.CONFIG_DISPATHER)
        def switch_features_handler(self, ev):
                datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        self.switches[datapath.id] = datapath
        self.mac_to_port.setdefault(datapath.id, {})
        self.flows.setdefault(datapath.id, {})
        #install table-miss flow entry
        match = parse.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto, OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 10, match, actions)
        match = parser.OFPMatch(tcp_src = srcport, tcp_dst = dstport, in_port = in_port, ip_proto = 6,eth_type =
0x0800)
        ipv4_dst=("10.0.0.5")
        actions = [parser.OFPActionOutput(ofp.OFPP_CONTROLLER,
        ofproto.OFPCMIL_NO_BUFFER)]
        self.add_flow(datapath, 10, match, actions)

        @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
        def _packet_in_handler(self, ev):
                 msg = ev.msg
    datapath = msg.datapath
```

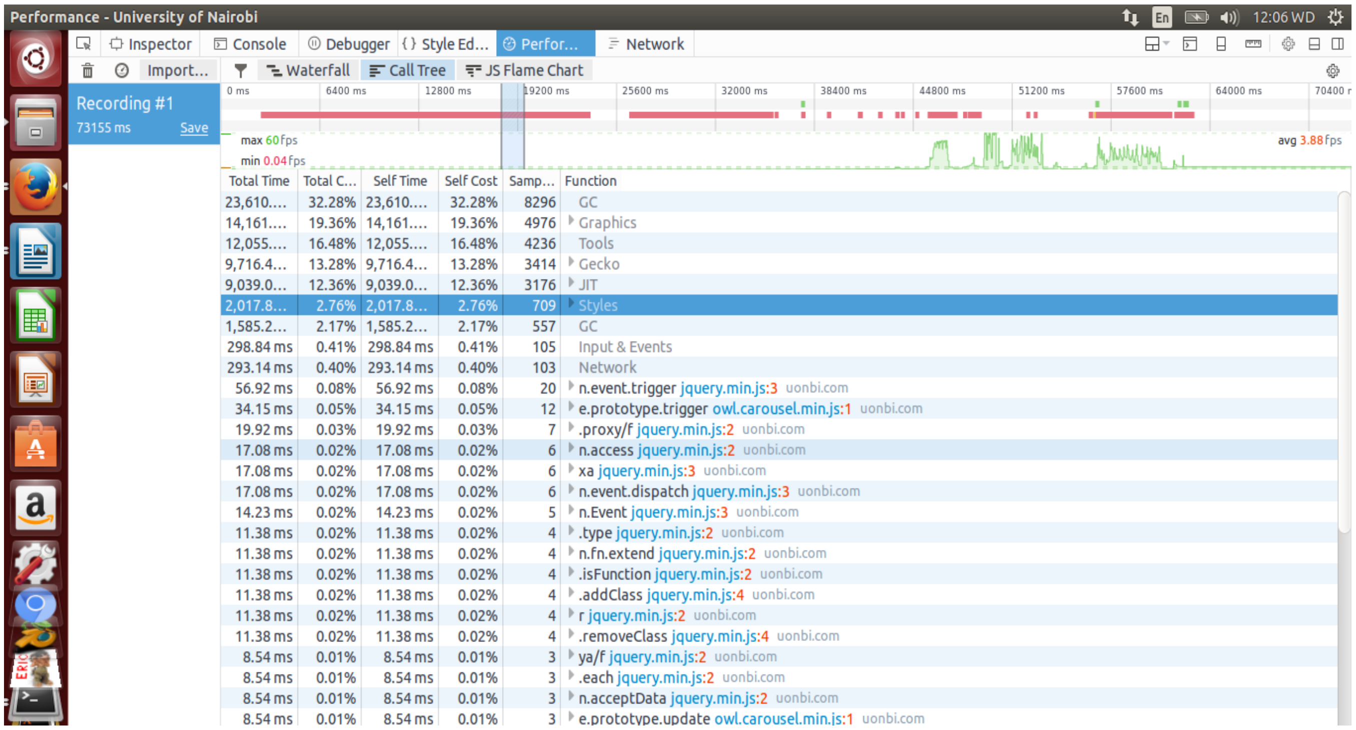
```
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        if (pkt.get_protocol(tcp.tcp) and eth.dst == "00:00:00:00:00:01"):
            #we get an unmatched TCP packet from h1
            tc = pkt.get_protocols(tcp.tcp)[0]
            dstport = tc.dst_port
            srcport = tc.src_port
            dst = eth.dst
            src = eth.src
            dpid = datapath.id
            self.logger.info("Received TCP packet\n srcport: %d\tdstport: %d",
                             srcport, dstport)
            match = parser.OFPMatch(tcp_src = srcport, tcp_dst = dstport, in_port = in_port, ip_proto = 6,eth_type =
0x0800)
            if self.usage["Port1"] < self.usage["Port3"]:
                self.logger.info("Pushing TCP flow to Port3")
                actions = [parser.OFPActionOutput(5)]
                self.add_flow(datapath, 20, match, actions)
                match = parser.OFPMatch(tcp_src = srcport, tcp_dst = dstport, in_port = 5, ip_proto = 6,eth_type =
0x0800)
                actions = [parser.OFPActionOutput(in_port)]
                self.add_flow(datapath, 20, match, actions) #Make reverse flow
            else :
                self.logger.info("Pushing TCP flow to Port4")
                actions = [parser.OFPActionOutput(4)]
                self.add_flow(datapath, 20, match, actions)
                match = parser.OFPMatch(tcp_src = srcport, tcp_dst = dstport, in_port = 5, ip_proto = 6, eth_type =
0x0800)
                actions = [parser.OFPActionOutput(in_port)]
                self.add_flow(datapath, 20, match, actions)

            data = None
            if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                data = msg.data
            out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                 in_port=in_port, actions=actions, data=data)
            datapath.send_msg(out)
        else:
            dst = eth.dst
            src = eth.src


            dpid = datapath.id
            self.mac_to_port.setdefault(dpid, {})
            self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

                                # learn a mac address to avoid FLOOD next time.
            self.mac_to_port[dpid][src] = in_port

            if dst in self.mac_to_port[dpid]:
                out_port = self.mac_to_port[dpid][dst]
            else:
                out_port = ofproto.OFPP_FLOOD

            actions = [parser.OFPActionOutput(out_port)]
            # install a flow to avoid packet_in next time
            if out_port != ofproto.OFPP_FLOOD:
                match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
                self.add_flow(datapath, 1, match, actions)

            data = None
            if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                data = msg.data

            out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions, data=data)
              datapath.send_msg(out)

class SimpleSwitchController(ControllerBase):
    def __init__(self, req, link, data, **config):
        super(SimpleSwitchController, self).__init__(req, link, data, **config)
        self.simpl_switch_spp = data[simple_switch_instance_name]

    @route('simpleswitch', url, methods=['GET'], requirements={'dpid': dpid_lib.DPID_PATTERN})
    def list_mac_table(self, req, **kwargs):
        simple_switch = self.simpl_switch_spp
        dpid = dpid_lib.str_to_dpid(kwargs['dpid'])

        if dpid not in simple_switch.mac_to_port:
            return Response(status=404)

##          mac_table = simple_switch.mac_to_port.get(dpid, {})
        body = json.dumps(mac_table)
        return Response(content_type='application/json', body=body)

    @route('simpleswitch', url, methods=['PUT'], requirements={'dpid': dpid_lib.DPID_PATTERN})
    def put_mac_table(self, req, **kwargs):

        simple_switch = self.simpl_switch_spp
        dpid = dpid_lib.str_to_dpid(kwargs['dpid'])
        new_entry = eval(req.body)

        if dpid not in simple_switch.mac_to_port:
            return Response(status=404)

        try:
            mac_table = simple_switch.set_mac_to_port(dpid, new_entry)
            body = json.dumps(mac_table)
            return Response(content_type='application/json', body=body)
        except Exception as e:
                                  print e
        return Response(status=500)
```

**II. Mininet Topology**

```
from mininet.cliimport CLI
from mininet.link import Link
from mininet.link import TCLink
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.term import makeTerm
if ' main ' == name :
```

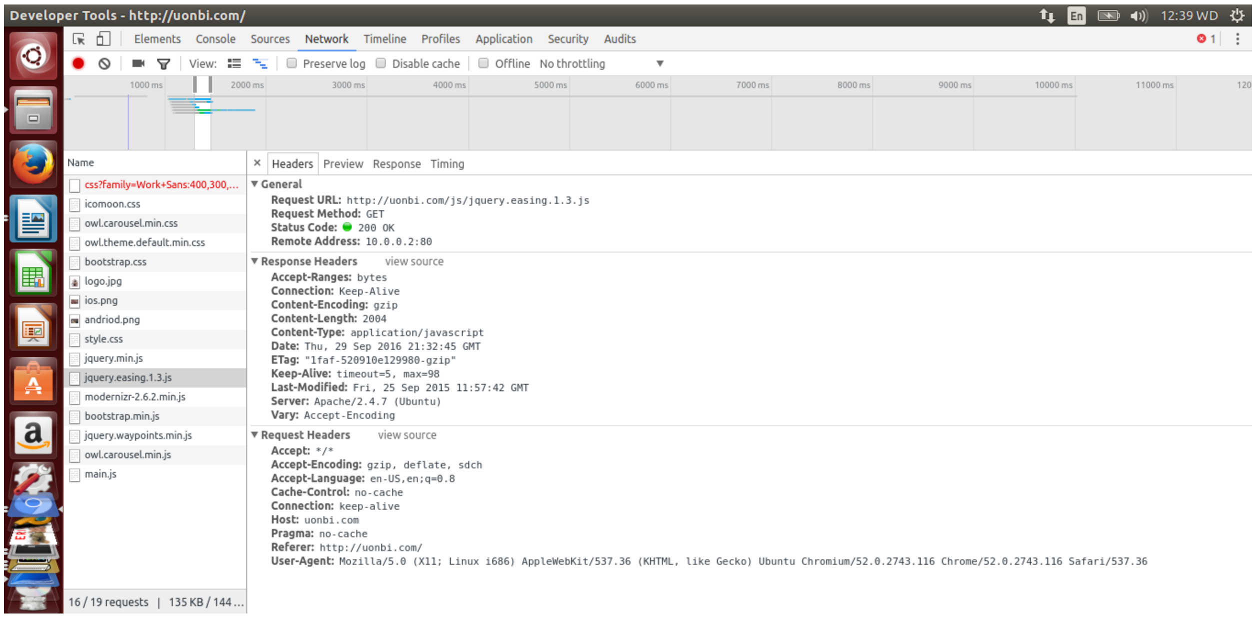
```
net = Mininet (controller=RemoteController,
autoStaticArp=True,autoSetMacs=True )
c0 = net.addController('c0 ')
s1 = net.addSwitch('s1')
h1 = net.addHost('h1')
h2 = net.addHost('h2')
h3 = net.addHost('h3')
h4 = net.addHost('h4')
h5 = net.addHost('h5')
TCLink(h1,s1,port2=1,bw=10,delay='10ms')
TCLink(h2,s1,port2=2,bw=1,delay='10ms')
TCLink(h3,s1,port2=3,bw=100,delay='10ms')
TCLink(h4,s1,port2=4,bw=10, de lay='10ms')
TCLink(h5,s1,port2=5,delay='10ms')
net.build()
c0.start()
s1.start([c0])
s1.sendCmd('ovs-vsctl set bridge s1 protocols=OpenFlow13')
CLI(net)
net.stop()
```

## Performance Measurement – Firefox



| Total Time | Total C... | Self Time | Self Cost | Samp... | Function |
|---|---|---|---|---|---|
| 23,610.... | 32.28% | 23,610.... | 32.28% | 8296 | GC |
| 14,161.... | 19.36% | 14,161.... | 19.36% | 4976 | Graphics |
| 12,055.... | 16.48% | 12,055.... | 16.48% | 4236 | Tools |
| 9,716.4... | 13.28% | 9,716.4... | 13.28% | 3414 | Gecko |
| 9,039.0... | 12.36% | 9,039.0... | 12.36% | 3176 | JIT |
| 2,017.8... | 2.76% | 2,017.8... | 2.76% | 709 | Styles |
| 1,585.2... | 2.17% | 1,585.2... | 2.17% | 557 | GC |
| 298.84 ms | 0.41% | 298.84 ms | 0.41% | 105 | Input & Events |
| 293.14 ms | 0.40% | 293.14 ms | 0.40% | 103 | Network |
| 56.92 ms | 0.08% | 56.92 ms | 0.08% | 20 | n.event.trigger jquery.min.js:3 uonbi.com |
| 34.15 ms | 0.05% | 34.15 ms | 0.05% | 12 | e.prototype.trigger owl.carousel.min.js:1 uonbi.com |
| 19.92 ms | 0.03% | 19.92 ms | 0.03% | 7 | .proxy/f jquery.min.js:2 uonbi.com |
| 17.08 ms | 0.02% | 17.08 ms | 0.02% | 6 | n.access jquery.min.js:2 uonbi.com |
| 17.08 ms | 0.02% | 17.08 ms | 0.02% | 6 | xa jquery.min.js:3 uonbi.com |
| 17.08 ms | 0.02% | 17.08 ms | 0.02% | 6 | n.event.dispatch jquery.min.js:3 uonbi.com |
| 14.23 ms | 0.02% | 14.23 ms | 0.02% | 5 | n.Event jquery.min.js:3 uonbi.com |
| 11.38 ms | 0.02% | 11.38 ms | 0.02% | 4 | .type jquery.min.js:2 uonbi.com |
| 11.38 ms | 0.02% | 11.38 ms | 0.02% | 4 | n.fn.extend jquery.min.js:2 uonbi.com |
| 11.38 ms | 0.02% | 11.38 ms | 0.02% | 4 | .isFunction jquery.min.js:2 uonbi.com |
| 11.38 ms | 0.02% | 11.38 ms | 0.02% | 4 | .addClass jquery.min.js:4 uonbi.com |
| 11.38 ms | 0.02% | 11.38 ms | 0.02% | 4 | r jquery.min.js:2 uonbi.com |
| 11.38 ms | 0.02% | 11.38 ms | 0.02% | 4 | .removeClass jquery.min.js:4 uonbi.com |
| 8.54 ms | 0.01% | 8.54 ms | 0.01% | 3 | ya/f jquery.min.js:2 uonbi.com |
| 8.54 ms | 0.01% | 8.54 ms | 0.01% | 3 | .each jquery.min.js:2 uonbi.com |
| 8.54 ms | 0.01% | 8.54 ms | 0.01% | 3 | n.acceptData jquery.min.js:2 uonbi.com |
| 8.54 ms | 0.01% | 8.54 ms | 0.01% | 3 | e.prototype.update owl.carousel.min.js:1 uonbi.com |

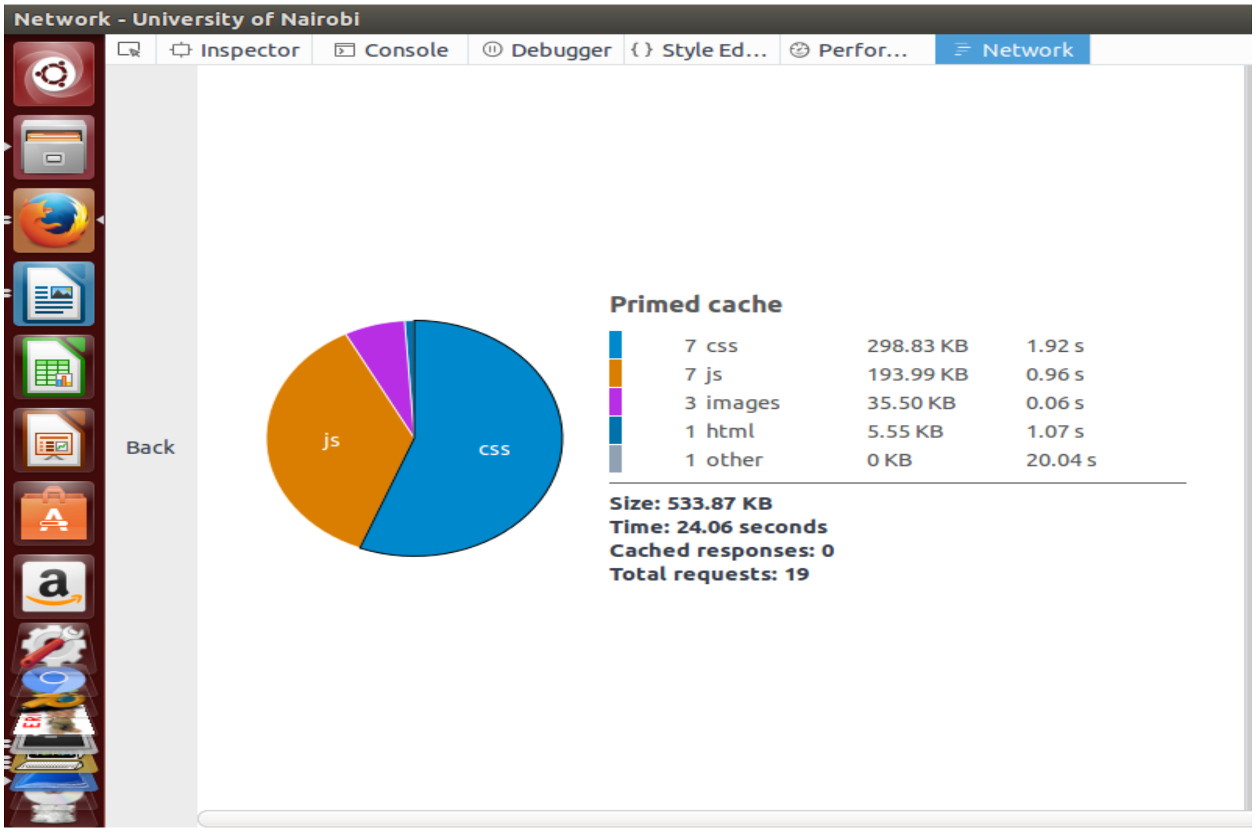
## Firefox – Get request


Developer Tools - http://uonbi.com/
Elements Console Sources Network Timeline Profiles Application Security Audits
View:
Preserve log
Disable cache
Offline
No throttling
1000 ms 2000 ms 3000 ms 4000 ms 5000 ms 6000 ms 7000 ms 8000 ms 9000 ms 10000 ms 11000 ms
Name
Headers Preview Response Timing

```
css?family=Work+Sans:400,300,...
icomoon.css
owl.carousel.min.css
owl.theme.default.min.css
bootstrap.css
logo.jpg
ios.png
andriod.png
style.css
jquery.min.js
jquery.easing.1.3.js
modernizr-2.6.2.min.js
bootstrap.min.js
jquery.waypoints.min.js
owl.carousel.min.js
main.js

16 / 19 requests | 135 KB / 144 ...
```

```
▼ General
    Request URL: http://uonbi.com/js/jquery.easing.1.3.js
    Request Method: GET
    Status Code: ● 200 OK
    Remote Address: 10.0.0.2:80
▼ Response Headers      view source
    Accept-Ranges: bytes
    Connection: Keep-Alive
    Content-Encoding: gzip
    Content-Length: 2004
    Content-Type: application/javascript
    Date: Thu, 29 Sep 2016 21:32:45 GMT
    ETag: "1faf-520910e129980-gzip"
    Keep-Alive: timeout=5, max=98
    Last-Modified: Fri, 25 Sep 2015 11:57:42 GMT
    Server: Apache/2.4.7 (Ubuntu)
    Vary: Accept-Encoding
▼ Request Headers      view source
    Accept: */*
    Accept-Encoding: gzip, deflate, sdch
    Accept-Language: en-US,en;q=0.8
    Cache-Control: no-cache
    Connection: keep-alive
    Host: uonbi.com
    Pragma: no-cache
    Referer: http://uonbi.com/
    User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/52.0.2743.116 Chrome/52.0.2743.116 Safari/537.36
```

## Cached Responses – Firefox

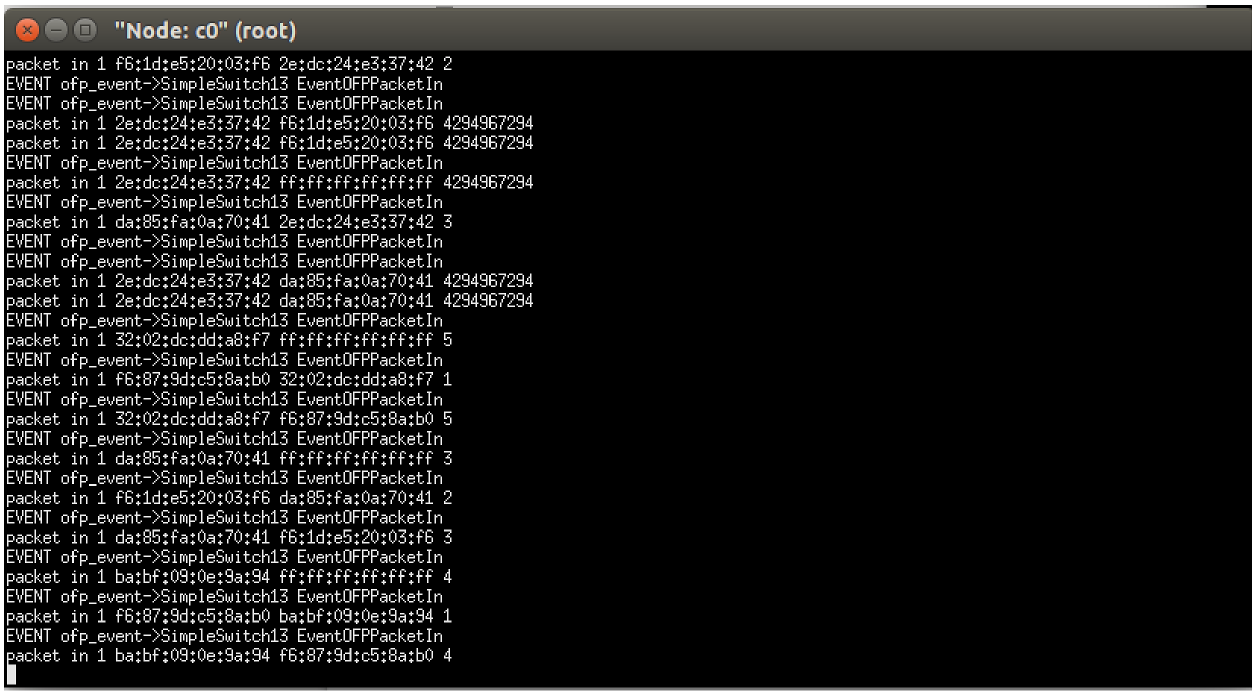


**Primed cache**

| | | | |
|---|---|---|---|
| 🟦 | 7 css | 298.83 KB | 1.92 s |
| 🟧 | 7 js | 193.99 KB | 0.96 s |
| 🟪 | 3 images | 35.50 KB | 0.06 s |
| 🟦 | 1 html | 5.55 KB | 1.07 s |
| ⬜ | 1 other | 0 KB | 20.04 s |

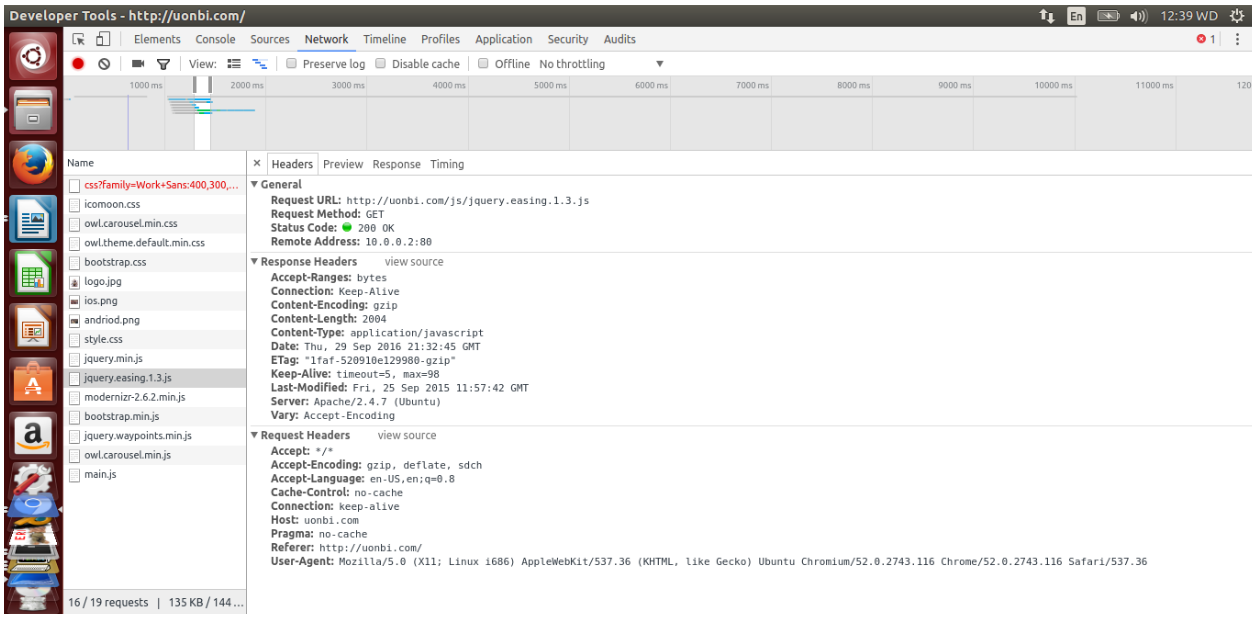
**Size: 533.87 KB**
**Time: 24.06 seconds**
**Cached responses: 0**
**Total requests: 19**

## Ryu Controller – Flows



```
packet in 1 f6:1d:e5:20:03:f6 2e:dc:24:e3:37:42 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 2e:dc:24:e3:37:42 f6:1d:e5:20:03:f6 4294967294
packet in 1 2e:dc:24:e3:37:42 f6:1d:e5:20:03:f6 4294967294
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 2e:dc:24:e3:37:42 ff:ff:ff:ff:ff:ff 4294967294
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 da:85:fa:0a:70:41 2e:dc:24:e3:37:42 3
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 2e:dc:24:e3:37:42 da:85:fa:0a:70:41 4294967294
packet in 1 2e:dc:24:e3:37:42 da:85:fa:0a:70:41 4294967294
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 32:02:dc:dd:a8:f7 ff:ff:ff:ff:ff:ff 5
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 f6:87:9d:c5:8a:b0 32:02:dc:dd:a8:f7 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 32:02:dc:dd:a8:f7 f6:87:9d:c5:8a:b0 5
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 da:85:fa:0a:70:41 ff:ff:ff:ff:ff:ff 3
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 f6:1d:e5:20:03:f6 da:85:fa:0a:70:41 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 da:85:fa:0a:70:41 f6:1d:e5:20:03:f6 3
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 ba:bf:09:0e:9a:94 ff:ff:ff:ff:ff:ff 4
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 f6:87:9d:c5:8a:b0 ba:bf:09:0e:9a:94 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 ba:bf:09:0e:9a:94 f6:87:9d:c5:8a:b0 4
```

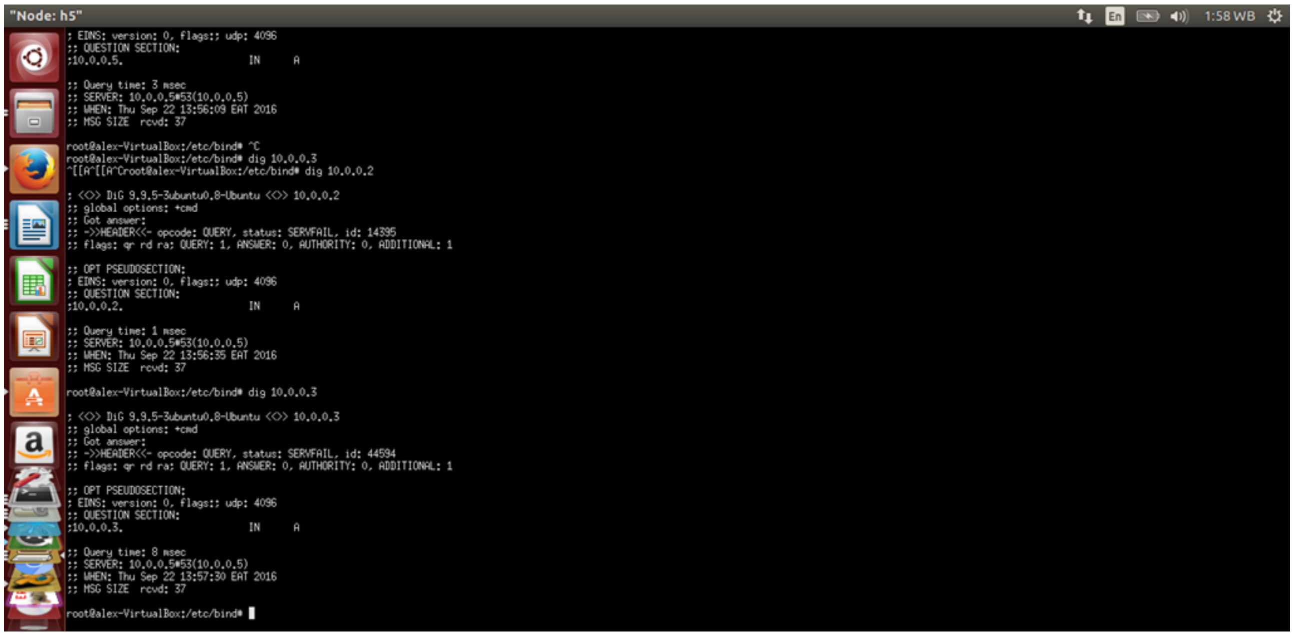
## Chrome – Get Request



```
css?family=Work+Sans:400,300,...
icomoon.css
owl.carousel.min.css
owl.theme.default.min.css
bootstrap.css
logo.jpg
ios.png
andriod.png
style.css
jquery.min.js
jquery.easing.1.3.js
modernizr-2.6.2.min.js
bootstrap.min.js
jquery.waypoints.min.js
owl.carousel.min.js
main.js
16 / 19 requests | 135 KB / 144 ...
```

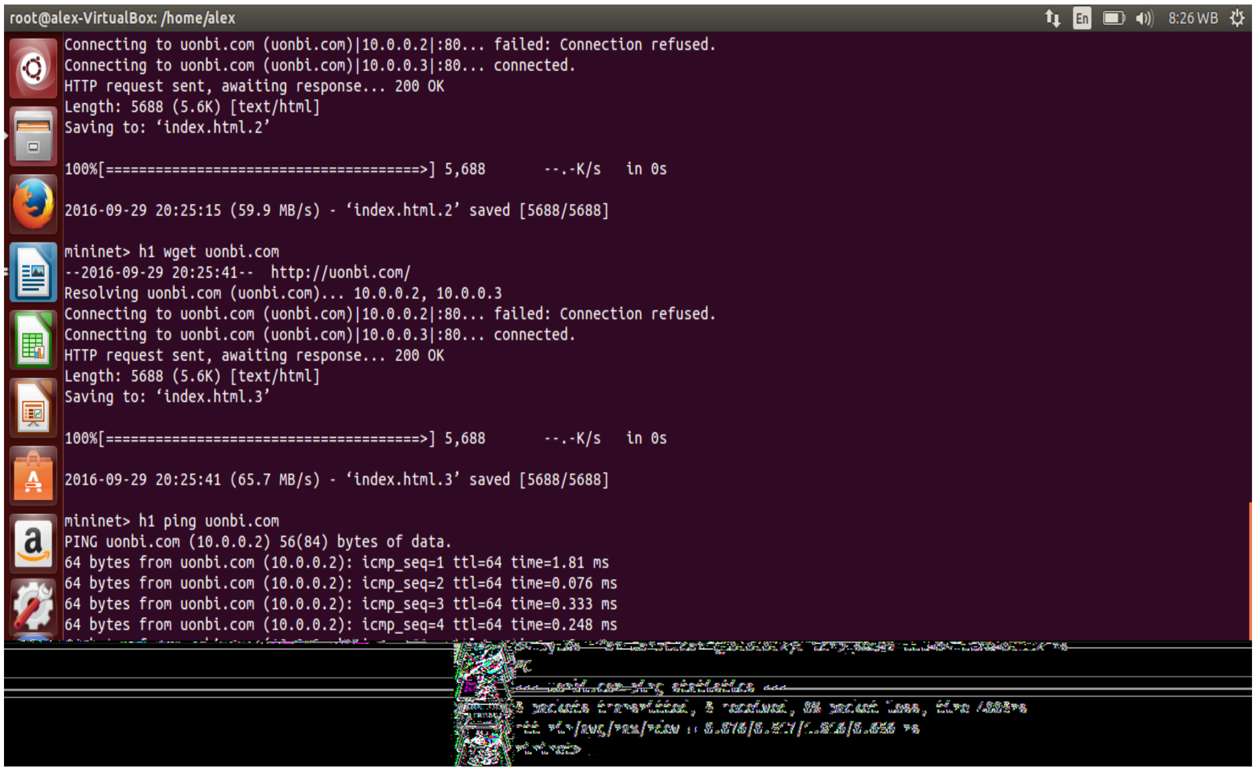
```
▼ General
    Request URL: http://uonbi.com/js/jquery.easing.1.3.js
    Request Method: GET
    Status Code: ● 200 OK
    Remote Address: 10.0.0.2:80
▼ Response Headers        view source
    Accept-Ranges: bytes
    Connection: Keep-Alive
    Content-Encoding: gzip
    Content-Length: 2004
    Content-Type: application/javascript
    Date: Thu, 29 Sep 2016 21:32:45 GMT
    ETag: "1faf-520910e129980-gzip"
    Keep-Alive: timeout=5, max=98
    Last-Modified: Fri, 25 Sep 2015 11:57:42 GMT
    Server: Apache/2.4.7 (Ubuntu)
    Vary: Accept-Encoding
▼ Request Headers        view source
    Accept: */*
    Accept-Encoding: gzip, deflate, sdch
    Accept-Language: en-US,en;q=0.8
    Cache-Control: no-cache
    Connection: keep-alive
    Host: uonbi.com
    Pragma: no-cache
    Referer: http://uonbi.com/
    User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/52.0.2743.116 Chrome/52.0.2743.116 Safari/537.36
```

## VI. Sample DNS Server  Name Resolution Statistics



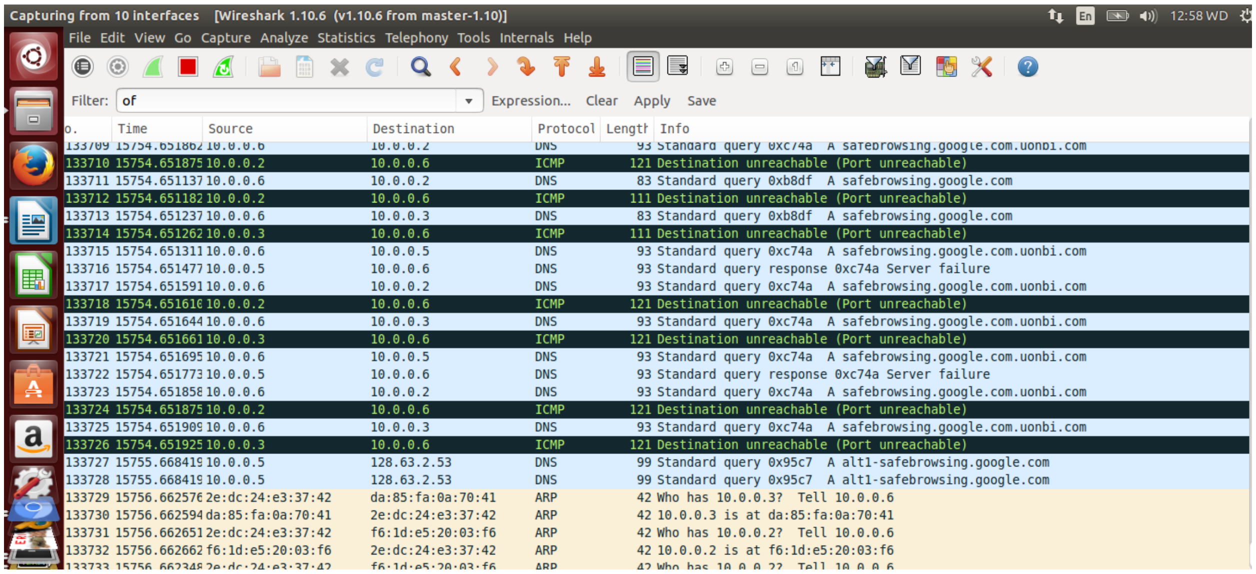
```
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;10.0.0.5.                      IN      A

;; Query time: 3 msec
;; SERVER: 10.0.0.5#53(10.0.0.5)
;; WHEN: Thu Sep 22 13:56:09 EAT 2016
;; MSG SIZE  rcvd: 37

root@alex-VirtualBox:/etc/bind# ^C
root@alex-VirtualBox:/etc/bind# dig 10.0.0.3
^[[A^[[A^Croot@alex-VirtualBox:/etc/bind# dig 10.0.0.2

; <<>> DiG 9.9.5-3ubuntu0.8-Ubuntu <<>> 10.0.0.2
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 14395
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;10.0.0.2.                      IN      A

;; Query time: 1 msec
;; SERVER: 10.0.0.5#53(10.0.0.5)
;; WHEN: Thu Sep 22 13:56:35 EAT 2016
;; MSG SIZE  rcvd: 37

root@alex-VirtualBox:/etc/bind# dig 10.0.0.3

; <<>> DiG 9.9.5-3ubuntu0.8-Ubuntu <<>> 10.0.0.3
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 44594
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;10.0.0.3.                      IN      A

;; Query time: 8 msec
;; SERVER: 10.0.0.5#53(10.0.0.5)
;; WHEN: Thu Sep 22 13:57:30 EAT 2016
;; MSG SIZE  rcvd: 37

root@alex-VirtualBox:/etc/bind# 
```

## IV. Mininet



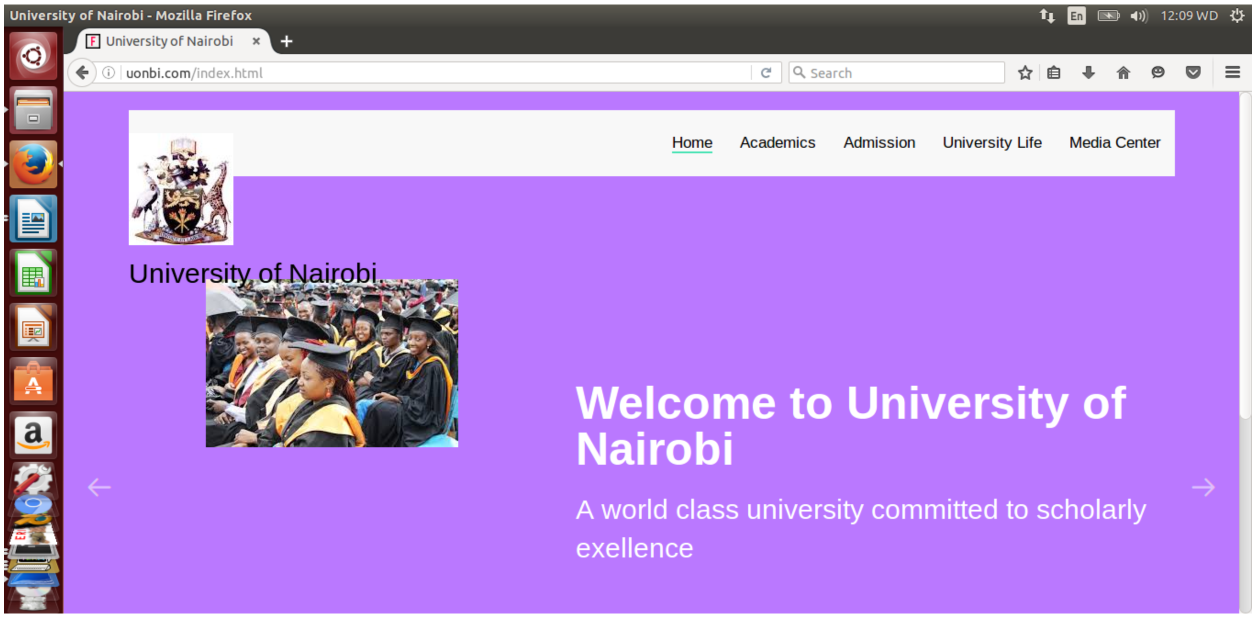
```
Connecting to uonbi.com (uonbi.com)|10.0.0.2|:80... failed: Connection refused.
Connecting to uonbi.com (uonbi.com)|10.0.0.3|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5688 (5.6K) [text/html]
Saving to: ‘index.html.2’

100%[=====================================>] 5,688       --.-K/s   in 0s

2016-09-29 20:25:15 (59.9 MB/s) - ‘index.html.2’ saved [5688/5688]

mininet> h1 wget uonbi.com
--2016-09-29 20:25:41--  http://uonbi.com/
Resolving uonbi.com (uonbi.com)... 10.0.0.2, 10.0.0.3
Connecting to uonbi.com (uonbi.com)|10.0.0.2|:80... failed: Connection refused.
Connecting to uonbi.com (uonbi.com)|10.0.0.3|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5688 (5.6K) [text/html]
Saving to: ‘index.html.3’

100%[=====================================>] 5,688       --.-K/s   in 0s

2016-09-29 20:25:41 (65.7 MB/s) - ‘index.html.3’ saved [5688/5688]

mininet> h1 ping uonbi.com
PING uonbi.com (10.0.0.2) 56(84) bytes of data.
64 bytes from uonbi.com (10.0.0.2): icmp_seq=1 ttl=64 time=1.81 ms
64 bytes from uonbi.com (10.0.0.2): icmp_seq=2 ttl=64 time=0.076 ms
64 bytes from uonbi.com (10.0.0.2): icmp_seq=3 ttl=64 time=0.333 ms
64 bytes from uonbi.com (10.0.0.2): icmp_seq=4 ttl=64 time=0.248 ms
[illegible]
^C
--- uonbi.com ping statistics ---
[illegible] packets transmitted, [illegible] received, 0% packet loss, time [illegible]ms
rtt min/avg/max/mdev = [illegible] ms
mininet>
```

## VI. Wireshark Statistics


Capturing from 10 interfaces [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Expression... Clear Apply Save
12:58 WD

```
o.     Time         Source            Destination       Protocol Length Info
133709 15754.651862 10.0.0.6          10.0.0.2          DNS        93 Standard query 0xc74a  A safebrowsing.google.com.uonbi.com
133710 15754.651875 10.0.0.2          10.0.0.6          ICMP      121 Destination unreachable (Port unreachable)
133711 15754.651137 10.0.0.6          10.0.0.2          DNS        83 Standard query 0xb8df  A safebrowsing.google.com
133712 15754.651182 10.0.0.2          10.0.0.6          ICMP      111 Destination unreachable (Port unreachable)
133713 15754.651237 10.0.0.6          10.0.0.3          DNS        83 Standard query 0xb8df  A safebrowsing.google.com
133714 15754.651262 10.0.0.3          10.0.0.6          ICMP      111 Destination unreachable (Port unreachable)
133715 15754.651311 10.0.0.6          10.0.0.5          DNS        93 Standard query 0xc74a  A safebrowsing.google.com.uonbi.com
133716 15754.651477 10.0.0.5          10.0.0.6          DNS        93 Standard query response 0xc74a Server failure
133717 15754.651591 10.0.0.6          10.0.0.2          DNS        93 Standard query 0xc74a  A safebrowsing.google.com.uonbi.com
133718 15754.651616 10.0.0.2          10.0.0.6          ICMP      121 Destination unreachable (Port unreachable)
133719 15754.651644 10.0.0.6          10.0.0.3          DNS        93 Standard query 0xc74a  A safebrowsing.google.com.uonbi.com
133720 15754.651661 10.0.0.3          10.0.0.6          ICMP      121 Destination unreachable (Port unreachable)
133721 15754.651695 10.0.0.6          10.0.0.5          DNS        93 Standard query 0xc74a  A safebrowsing.google.com.uonbi.com
133722 15754.651773 10.0.0.5          10.0.0.6          DNS        93 Standard query response 0xc74a Server failure
133723 15754.651858 10.0.0.6          10.0.0.2          DNS        93 Standard query 0xc74a  A safebrowsing.google.com.uonbi.com
133724 15754.651875 10.0.0.2          10.0.0.6          ICMP      121 Destination unreachable (Port unreachable)
133725 15754.651909 10.0.0.6          10.0.0.3          DNS        93 Standard query 0xc74a  A safebrowsing.google.com.uonbi.com
133726 15754.651925 10.0.0.3          10.0.0.6          ICMP      121 Destination unreachable (Port unreachable)
133727 15755.668419 10.0.0.5          128.63.2.53       DNS        99 Standard query 0x95c7  A alt1-safebrowsing.google.com
133728 15755.668419 10.0.0.5          128.63.2.53       DNS        99 Standard query 0x95c7  A alt1-safebrowsing.google.com
133729 15756.662576 2e:dc:24:e3:37:42 da:85:fa:0a:70:41 ARP        42 Who has 10.0.0.3?  Tell 10.0.0.6
133730 15756.662594 da:85:fa:0a:70:41 2e:dc:24:e3:37:42 ARP        42 10.0.0.3 is at da:85:fa:0a:70:41
133731 15756.662651 2e:dc:24:e3:37:42 f6:1d:e5:20:03:f6 ARP        42 Who has 10.0.0.2?  Tell 10.0.0.6
133732 15756.662662 f6:1d:e5:20:03:f6 2e:dc:24:e3:37:42 ARP        42 10.0.0.2 is at f6:1d:e5:20:03:f6
133733 15756.662348 2e:dc:24:e3:37:42 f6:1d:e5:20:03:f6 ARP        42 Who has 10.0.0.2?  Tell 10.0.0.6
```

## XI. Uonbi.com User Redirection test page



University of Nairobi

Welcome to University of Nairobi

A world class university committed to scholarly exellence