2016

Seamless Application Delivery Using Software Defined Exchanges

Divyashri Bhat
University of Massachusetts Amherst

SEAMLESS APPLICATION DELIVERY USING SOFTWARE DEFINED EXCHANGES

A Thesis Presented

by

DIVYASHRI BHAT

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

February 2016

Electrical and Computer Engineering

SEAMLESS APPLICATION DELIVERY USING SOFTWARE DEFINED EXCHANGES

A Thesis Presented

by

DIVYASHRI BHAT

Approved as to style and content by:

Michael Zink, Chair

David Irwin, Member

Tilman Wolf, Member

C.V.Hollot, Department Head
Electrical and Computer Engineering

ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to my advisor, Professor Michael Zink, who has the attitude and the substance of a genius and who inspired my interest in computer networking. Without his guidance and persistent help this thesis would not have been possible. He always encourages me and inspires me with lots of valuable insights and ideas. I am heartily thankful to Professor David Irwin and Professor Tilman Wolf for their constructive advice and invaluable help in my research and future career.

I would like to acknowledge the associated members of the GENI Project office at Raytheon BBN Technologies for their support in formulating ideas and experiments that are a major part of this work. In particular, I want to thank Niky Riga, Joe Mambretti, Russ Clark, Inder Monga, Jim Chen, Cas D'Angelo, Ron Hutchins and Chin Guok who helped me put together the resources used in this project. A special thank you also goes out to everyone in the Data Analytics Lab for the knowledge and experience they have shared with me.

Finally, I appreciate all of the sincere support from my family and my friends, this thesis pales in comparison to what I gained from them.

ABSTRACT

SEAMLESS APPLICATION DELIVERY USING SOFTWARE DEFINED EXCHANGES

FEBRUARY 2016

DIVYASHRI BHAT

B.E, BANGALORE INSTITUTE OF TECHNOLOGY, BANGALORE, INDIA

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Prof. Michael Zink

One of the main challenges in delivering content over the Internet today is the absence of a centralized monitoring and control system [38]. Software Defined Networking has paved the way to provide a much needed control over network traffic. OpenFlow is now being standardized as part of the Open Networking Foundation, and Software Defined Exchanges (SDXes) provide a framework to use OpenFlow for multi-domain routing. Prototype deployments of Software Defined Exchanges have recently come into existence as a platform for Future Internet Architecture to eliminate the need for core routing technology used in today's Internet. In this work, we look at how application delivery, in particular, Dynamic Adaptive Streaming over HTTP (DASH) and Nowcasting take advantage of a Software Defined Exchange. We compare unsophisticated controllers to more sophisticated ones which we call a "load balancer" and find that implementing a good controller for inter-domain routing can result in better network utilization and application performance. We then design, develop and evaluate a prototype for a Content Distribution Network (CDN) that uses resources at SDXes to provide higher quality bitrates for a DASH client.

TABLE OF CONTENTS

[1]This work is to appear at Infocomm 2016 and was done in collaboration with Zdravko Bozakov (Leibniz Universität Hannover) and Amr Rizk (University of Massachusetts, Amherst)

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Introduction

In recent years, Software Defined Networks have emerged as a preferred deployment in various types of networks such as data centers [58], Wide Area Networks (WANs) [29], and Wireless Networks [55] [35]. Unlike traditional Internet Protocol (IP-based) networks, SDNs allow separation of the control and data planes and thus, provide network administrators and applications with the ability to implement dynamic routing based on feedback from the network or application. They also provide the flexibility to implement network control and traffic forwarding in either distributed [59] or centralized setups [60].

Internet Exchange Points (IXPs) have been a fundamental part of the Internet Architecture for many years. A typical view of IXPs is that of a physical switch administered by a third party that allows ISPs to connect their routers to the switch. Routing in IXPs is principally governed by BGP routing policies such as longest prefix matching [47]. According to PeeringDB [6], there are currently 510 public exchange points with a total of 6480 peering networks and traffic levels for each AS ranging from 20Mbps to about 1Tbps and these numbers can only be expected to increase in the coming years, which compels the study of SDXes as a network entity for implementing sophisticated policies that go far beyond BGP routing.

IXPs have harmoniously co-existed with the Internet until now. We believe that with some enhancements they are capable of elevating networking capabilities of the Internet by manifolds. Currently, the research community is discussing Software Defined Exchanges (SDXes) as an enhanced form of IXPs, which are of particular interest as they leverage the advantages of Software Defined Networks to allow dynamic traffic steering, peering policies between network providers and applications and trusted third party policy implementation. Gupta et al. [27] have presented SDXes as an alternative to IXPs and evaluated the advantages of implementing dynamic BGP routing in an OpenFlow switch that uses Pyretic [40] to implement dynamic policies.

Seamless content delivery continues to be a point of concern for several entities including ISPs and content providers with respect to network management and application delivery across various domains [23]. In a content-dominated Internet, a key challenge faced by content providers and ISPs is not only providing the client with a consistent content quality but also the highest possible content quality. In this work, we present our view of SDXes as a cloud consisting of IXPs, compute and storage which are all managed by a Software Defined framework and are particularly interested in understanding how an SDX-enabled network can benefit application transfer both in terms of providing better Quality of Service (QoS) and saving network bandwidth. As a preliminary analysis we evaluate the performance of a combination of two OpenFlow-based SDXes used to switch traffic between three independent domains (SDN and non-SDN) and used controllers to route traffic of two real-world applications across the SDXes. In our analysis, presented in Section 3, we considered two applications with different network requirements which are:

1) Nowcast - This is a weather based application where data is collected from radars and moved to a middle box for processing after which the image is displayed on a web page. This application requires high-bandwidth and low end-to-end latency in addition to having in-network compute and storage units.

2) VLC DASH - Dynamic Adaptive Streaming over HTTP (DASH) implemented in VLC is used to represent videos in segments of different qualities. The quality of the video delivered to a client is measured in single and multiple domains to analyze the advantages of using SDXes for such an application. This application requires high and consistent bandwidth between server and client while being able to handle a large number of HTTP requests. We use lessons learned from this prototype to design a framework for SDXes that can seamlessly integrate with any application to exploit the traffic engineering capabilities of SDN.

In this work, we develop and test a prototype REST interface front-end using MongoDB as the distributed database back-end, which integrates with an SDN controller to provide higher quality bitrates for the VLC DASH application. The rest of this paper is organized as follows. Section 2 contains a discussion of relevant work done by other researchers in this area followed by our initial SDX setup for analytical measurements described in Section 3. The architecture of our REST API framework for improving video delivery using SDXes is presented in Section 4, followed by a description of our testbed setup and experiment scenarios in Section 5. In Section 6, we present results for the basic evaluation of our prototype, including an interesting server-selection algorithm for a virtual CDN and then evaluate a real-world network path computation use case for our framework. We conclude this paper with Section 7 that includes a discussion of the possible advantages of SDXes as a Future Internet Architecture given our prototype design for a CDN architecture.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Software Defined Networks

Software Defined Networks (SDNs) were initially designed to provide better control and manageability for network operators. OpenFlow [39] is one of the most popular implementations of SDN and has been deployed in a production network by Google [29] as a large-scale Wide Area Network (WAN). They specifically mention the performance advantages of using OpenFlow for network programmability and applications. Two of the most prominent large-scale deployments of OpenFlow for research and education include the GENI Testbed [17] and OFELIA [56]. There has been considerable effort to better understand the growth in Network Function Virtualization that led to SDN deployment [31], [5]. In [22], Feamster et al. discuss myths and misconceptions regarding programmable networks and how SDN differs from network virtualization. They present OpenFlow as an outstanding example that allows traffic engineering based on 13 different packet headers which provides flexibility and improves manageability of large-scale networks.

During the last couple of years, researchers have looked into implementing routing policies using SDN. Bennesby et al. present their work on inter-domain routing using SDN in [16]; in particular, their work consists of time-of-convergence measurements for dynamic path reconfiguration of BGP protocols that were evaluated using Mininet [33]. OSHI [50] describes a hybrid IP/SDN routing model that uses OSPF implemented in Quagga and defines OpenvSwitch routes based on VLAN tags. In [20], Chetty and Feamster present the advantages of developing an interface that interacts with the underlying infrastructure in a home network. They show that SDN can complement existing functionality in a home network by providing users with additional information regarding their Internet Service Provider (ISP) and about applications that could potentially cause performance degradation.

2.2 Internet Exchange Points (IXP)

The main function of Internet Exchange Points is to serve as peering points for Autonomous Systems (AS). IXPs now implement peering policies between Content Distribution Networks (CDN) and ISPs to efficiently distribute content to clients [4] [1]. We believe that such powerful networking infrastructure should be enhanced further by co-locating the cloud with IXPs and most of our work focuses on leveraging this capability for application delivery. Chatzis et al. [19] present a survey of research and capabilities that involve IXPs and use traffic statistics to show that IXPs are an integral part of the Internet infrastructure. Gupta et al. are among those who believe that combining SDN capabilities with IXPs or SDXes will provide IXPs with considerably better manageability. In [27], they present a deployment of an SDX with policies implemented in Pyretic and show that using SDXes would reduce network component failure response time by magnitudes, allow seamless live VM migration between domains and allow third-party policy implementation. Cardigan [54] which is an implementation of the RouteFlow [49] project describes a real deployment of a "distributed routing fabric" between the Wellington Internet Exchange (WIX) and the Education Advanced Network of New Zealand (REANNZ). The authors of this paper demonstrate results of a real-deployment with actual ISPs peering at the SDXes but using an actual peering point constrains them from more fine-grained performance analysis in order to avoid interfering with real network traffic.

2.3 Application Aware Routing in SDN

PolicyCop [15] is recent work that implements and enforces third-party QoS based policies in SDN routing by incorporating a management plane in conjunction with the control and data plane already implemented by SDN. In a previous work, the authors describe Payless [21] which is a network monitoring software for OpenFlow-based networks and allows better network administration through the use of RESTful APIs.

Research on application aware routing focuses on providing users with a better Quality of Experience (QoE). Jarschel et al. [30] have examined the benefits of using SDN in conjunction with Deep Packet Inspection (DPI) for enhancing the QoE for YouTube users. A more recent work [62] on application aware SDN routing looks at resource management by dynamically allocating network resources based on application requirements. Their experiments look at queuing strategies for flows in the OpenFlow protocol and shows that flow queues can be prioritized for better traffic management but TCP traffic could suffer short-time performance degradation.

Although the work described in this section is similar to ours in that it looks into evaluating application aware routing in SDNs, the networks they use for evaluation co-exist with the Internet and therefore, do not consider end-to-end performance isolation. In addition, we evaluate several applications such as video streaming applications and a short-term weather prediction application on a real SDX-enabled GENI testbed in order to gain insight into incorporating SDXes as a part of Future Internet Architecture.

After the emergence of OpenFlow, there has been some work that investigates the use of a control plane for improving video delivery in a CDN. One work considers the use of a control plane for load-balancing in a CDN to improve the QoE of live video delivery [41]. The authors design, implement and evaluate a DNS load balancing system with a hybrid (distributed and centralized) control system for live video streaming. Our work differs from this in that we use OpenAPIs via a REST framework for content distribution and OpenFlow for controller implementation, which is easy to integrate with any client application. Ganjam et al. [24] have looked at using co-ordinated control plane for routing video in the Internet. Their work describes a client based monitoring and control system where the client is supposed to make intelligent decisions based on the information gathered by the monitoring system. Unlike our system, this architecture is restricted to a single CDN deployment where decisions have to be made on a per client basis. This leads to considerable overhead for the control plane that cannot be ignored. In our system, we use a similar deployment without client-based granularity but achieve high quality bitrates without this additional overhead.

CHAPTER 3

A MEASUREMENT STUDY OF APPLICATION DELIVERY USING SOFTWARE DEFINED EXCHANGES (SDXes)

To evaluate our hypothesis that envisions SDXes as a CDN, we present in the following sections a preliminary study of a virtual network with actual SDXes and use the two applications, Nowcast and VLC DASH, mentioned in Section 1.1 to evaluate the advantages of using traffic engineering approaches when compute and storage is co-located in a virtual network. We give a detailed description of our SDX setup and experiments carried out to understand the usefulness of SDXes, which enables us to design, implement and test our REST API framework as described in the rest of this work.

3.1 Architecture

This section gives a detailed description of our experimental setup. The resources used in this experiment were all reserved using GENI Aggregate Manager APIs [17]. The GENI network is deeply programmable and its resources consist of a combination of racks and layer-2 VLANs between campus networks across the world. Our architecture in GENI mainly consists of the following three parts.

3.1.1 SDXes

The SDXes are located at:

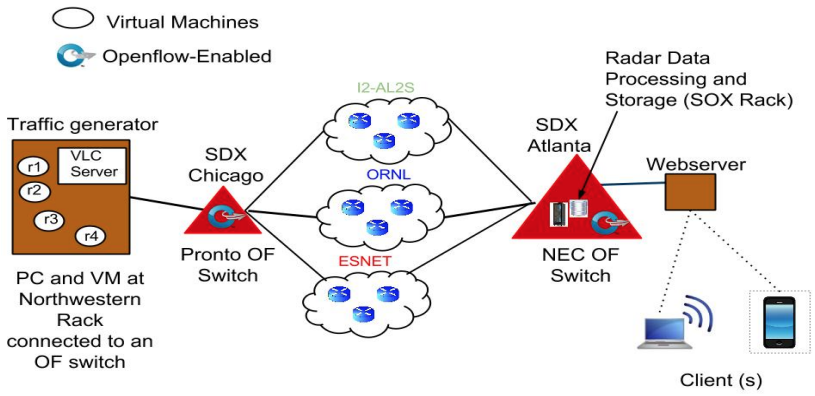


Figure 3.1: SDX Topology

Starlight (Chicago) - The Starlight SDX is a software defined implementation of an Internet Exchange Point which provides both international and national communications exchange. The Starlight SDX is currently a part of the Global Lamba Integrated Facility (GLIF) which is heavily involved in optical networking and research and provides a platform for researchers to use upto 100GB links for experimentation purposes [36].

Southern Crossroads (SoX - Atlanta) - The SoX infrastructure connects campus networks using Internet2 and allows programmability using FOAM/FlowVisor APIs that use OpenFlow. The SDX at SoX is designed to allow application-specific peering where multiple flow tables can be implemented with BGP as the default routing policy.

3.1.2 Application Software

Nowcast - The actual Nowcast application scenario consists of four radars in a grid system which send data to a middle box for processing. In our setup, the radar data is replayed from four (Virtual Machines) VMs in the GENI rack at Northwestern and is sent to a raw PC located in the SoX rack for processing and image generation. The image is then sent to a VM in the GENI rack at GeorgiaTech to be displayed on a web server there. The SDXes described above allows us to switch flows to different domains based on the available bandwidth in each domain.

VLC DASH - Dynamic Adaptive Streaming over HTTP (DASH) is a widely implemented video streaming approach, where a single video is separated into segments of equal length and each segment is represented in various qualities [53]. The application, more commonly, the client requests each segment in a quality that is computed by a DASH Engine and could be based on adaptation schemes such as Rate-based or Buffer-based. In a Rate-based adaptation scheme, the client requests the next segment in a quality based on the download rate of the current segment [42]. In a buffer-based adaptation scheme, the client uses the amount of data downloaded into a client buffer to estimate the quality of the next segment as shown in [28]. We use a raw PC in the Northwestern rack to host an Apache2 web-server with a DASH Dataset [34]. A single client is started on a PC in the SoX rack (this PC also acts as the process box for the Nowcast application).

3.1.3 Domains

Internet2 (I2) - The AL2S or Advanced Layer-2 Service Domain which is a part of the Internet2 network [2] provides a direct VLAN between an NEC OpenFlow Switch at SoX and the Starlight Pronto Switch in Chicago. This is Domain 1 shown in the Figure 3.1.

Oakridge National Lab (ORNL) - ORNL is a large data science research center and this network was initially setup to support research and experimentation at ORNL, Tennessee. They provide a VLAN between SoX and Starlight which is denoted by Domain 2 in Figure 3.1. All of these domains provide us with network speeds upto 100Gbps.

Energy Sciences Network (ESnet) - ESnet is a large scale national network which supports big data experiments in research and experimentation. They provide a VLAN which is depicted in Figure 3.1 as Domain 3.

3.2 Experiments

This section describes our experimental setup and scenarios.

3.2.1 Traffic Routing using OpenFlow

Here we describe different controllers that we use to route application traffic for our experiments. We use Trema [9], a Ruby-based tool, as the preferred API for implementing our OpenFlow controller. Trema provides us with the ability to use in-built modules such as the "Learning Switch" for simple port-mapping that works by flooding all interfaces of the switch when the first packet of a flow reaches the controller. We run this controller in the Georgia Tech rack, which also hosts the web server as described in 3.2.2.2. The learning switch controller stores interface-MAC address mapping for three switches in our setup, namely, the NEC OpenFlow at SoX-SDX, SoX rack switch and the GaTech rack switch.

3.2.1.1 Timer-based Path Switching

This is a simple controller that switches flows to a new domain in a round-robin fashion with a time-interval that is pre-determined at the time of running the controller. Although we tried varying time intervals, in this work we present results only for time intervals of 30s and 60s because the general performance trend of all timer-based controllers is similar to the ones shown here. This controller merely switches flows without taking into account any network parameters or performing any packet analysis and is non-reactive. We look at the performance of the VLC-DASH application while we run this controller as the video quality is quantifiable and can be measured using the Decision Rate for this application.

3.2.1.2 Throughput-based Path Switching (Load Balancer)

This controller is more sophisticated than the one described above and considers network parameters before selecting a path for a flow. This controller examines the statistics collected from the SDX switch at Starlight to extract the throughput available on each of the three domains and then switches a new flow to the least congested path (highest available throughput). Here, the throughput per domain is the cumulative aggregate of the instantaneous throughput on the domains after the controller is started. We also run this experiment by considering only the instantaneous throughput as opposed to the cumulative throughput but have omitted those results in this work as we obtain better performance when we consider the cumulative throughput.

3.2.2 Applications

The applications we use in our experiments are:

3.2.2.1 VLC-DASH

This application is a VLC implementation of Dynamic Adaptive Video Streaming over HTTP (DASH) [53] which is the representation of a video encoded in a standard format, i.e., H264 in segments having different bitrates. The VLC client is pre-configured to request 2s, 4s, 6s, 8s or 10s segments as desired by the application and each of these segments is available in five different qualities which are 240p, 360p, 480p, 720p and 1080p. Before requesting a segment, the VLC client measures the bandwidth from the server and then, requests the quality which best represents the available bandwidth from the server. The Decision Rate of the video is the bitrate of the video stream that the client requests and is quality of the segment that is served to the client. In our experiments, the client is configured to request segments of the default length which is 2s and the video server is Apache2 with a pre-loaded DASH dataset [34]. The sample video used in this case is 10 minutes long.

3.2.2.2 Nowcasting

Nowcasting is a short-term weather prediction application that allows emergency responders to perform timely evacuation in case of severe weather such as tornados, flash floods and hurricanes. The system consists of four radars that send measurements at predetermined intervals to a processing unit which generates an image to indicate severity of weather over a particular region. In our experiments, we replay previously collected measurements from four radars in Oklahoma using four VMs named r1, r2, r3 and r4 as shown in Figure 3.1 from the Northwestern GENI rack. The Nowcast processing algorithm runs on a bare metal machine in the SoX rack. An Apache2 server that hosts the generated image is deployed on a Xen virtual machine in the Georgia Tech rack. For the purpose of our measurements, we do not take into account the network between the nowcast processing and the Apache2 server because they are geographically situated close to each other.

3.2.2.2.1 Iperf We use Iperf [10] merely to generate competing TCP cross traffic between the radar VMs and the process box.

3.2.3 Experiment Scenarios

Here, we describe the steps we use to run different experiments with the objective of comparing the performance of applications transported using a sophisticated, reactive controller with that of a naïve, timer-based approach. In order to motivate our approach, our initial experiments are more focussed on independent domains carrying application traffic so that we can clearly show the advantages of using a multi-domain SDX approach.

3.2.3.1 Experiment 1: Single Domain Single Application

We start the VLC DASH client in the SoX rack and use each domain, i.e, I2-AL2S, ORNL and ESNET, to independently transport VLC DASH segment requests from the SoX Rack to the web server at Northwestern and the video streams from the web server back to the VLC DASH client. In addition, we start three TCP Iperf flows from each of the radar VMs to the SoX rack to serve as competing TCP traffic. We collect measurements for the bitrate requested by the client for each segment to analyze the video quality boost and degradation with respect to time for each domain. We also collect statistics from the SDX switch at Starlight to look at the total bandwidth utilization for each individual domain.

3.2.3.2 Experiment 2: Multi Domain Single Application

We repeat the experiment in 3.2.3.1 but this time we use all three domains to transport application traffic and allow our Trema controller to switch between domains every 30s and 60s respectively. Following this we run experiments with the load balancer controller that switches flows based on the aggregate throughput of each domain as seen from the Starlight SDX switch. We also repeat this experiment for the Nowcast experiment described in 3.2.2.2

3.2.3.3 Experiment 3: Multi Domain Multi Application

In this case, we collect measurements for each controller described in 3.2.1 while running VLC DASH and Nowcast applications simultaneously. As in the case of the experiments described in 3.2.3.1 and 3.2.3.2, we use LabWiki [45], an instrumentation and measurement tool provided by GENI, to start the VLC DASH client, measure the Decision Rate in case of the VLC DASH and collect total flow statistics for both applications. However, our results are only presented for the 30s-timer based controller in comparison with the load balancer controller 3.3.3 as the performance of the 30s timer is better than that of the 60s timer-based one.

3.2.4 Orchestration and Measurement

For running our VLC DASH client and collecting measurements for all our experiments we make use of a tool called LabWiki which was developed at NICTA and is specifically designed to orchestrate network-related experiments using Ruby-based scripts (OEDL). LabWiki is a web-interface for the Orbit Measurement Framework (OMF) [46] and it also allows us to view live visualisations of our experiment.

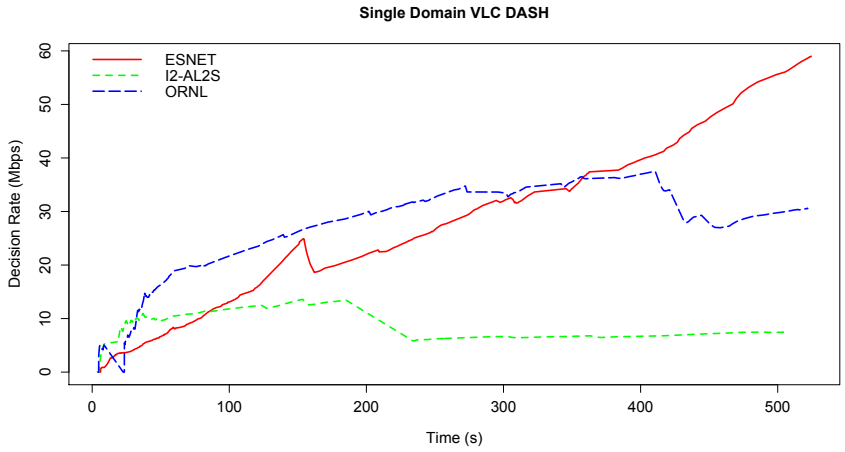
(a) **VLC DASH client** - Our VLC DASH client is started using a LabWiki script and we measure the Decision Rate as described in 3.2.2.1 over time.

(b) **OpenFlow Statistics Application** - Here we use LabWiki to collect measurements of the statistics obtained from the switch, such as cumulative throughput and instantaneous throughput with respect to time. We also collect long-term results to observe the controller performance over time.

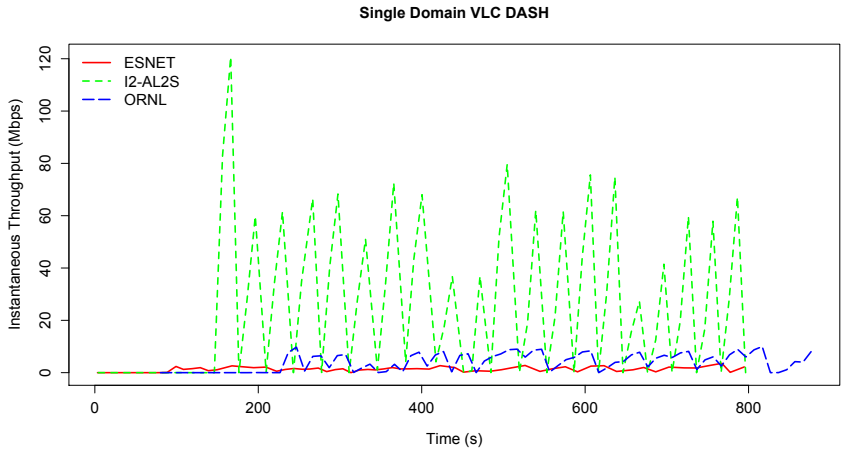
(c) **Nowcast Application** - Nowcast developed by researchers of the CASA project is a Linux-based application which uses the Local Data Manager [11] to start a listener on the process box and clients on the radar VMs. We use this application to evaluate the suitability of an SDX-enabled network for data-intensive applications. The performance is measured using 3.2.4 (b) described above.

3.3 Results

In this section, we present the results of the experiments as described in Section 3.2 with performance analysis for both the VLC and Nowcast applications.



(a)



(b)

Figure 3.2: Single Domain VLC DASH

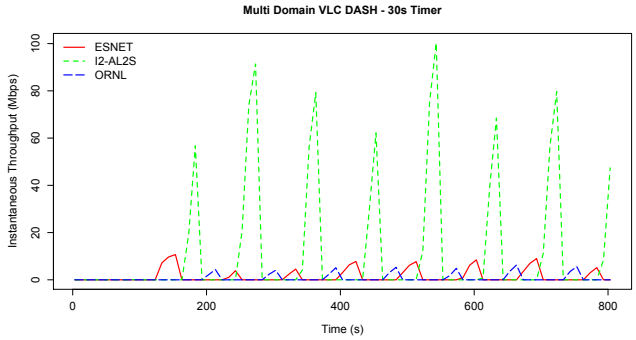
3.3.1 Single Domain Performance

In Figure 3.2(a), we present the results we obtain while running a VLC DASH client application along with three parallel TCP Iperf flows where we start the VLC client about 30s after we start the Iperf transmissions. Here, we see that the ORNL link provides a constant VLC DASH Decision Rate. ESNET also performs reasonably well and allows the client to request the highest available bitrate from the HTTP DASH server. However, the worst performance we see is for the Internet2-AL2S (I2-AL2S) link because not only does it take the longest time to converge to a constant bitrate but the client also does not request the highest available bitrate. We compare this with the results presented in Figure 3.2(b), which shows throughput measured at the SDX at Starlight. Figure 3.2(b) clearly shows that the I2-AL2S SDN domain has the highest throughput. We assume that there is some performance degradation happening on the path between the Starlight and the SoX SDX and will investigate this in future work.

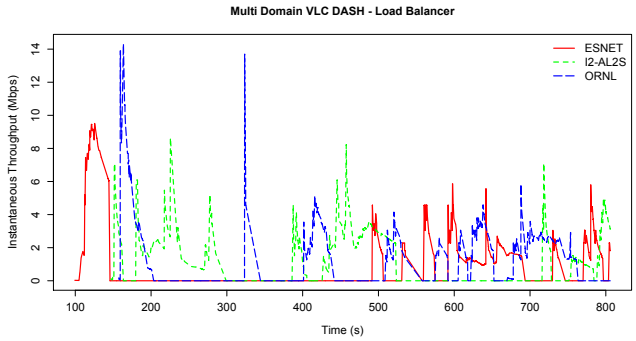
This initial measurement results clearly motivates the need for an environment that allows researchers to instrument SDXes to better analyze their behavior and measure performance.

3.3.2 Multi Domain Single Application Performance

Figure 3.3 shows the corresponding flow statistics results for two alternative controller implementations. In the first one, Figure 3.3(a), the path is switched every 30s and we see that the average throughput of the Internet2-AL2S path is the highest. The throughput of the ESNET and ORNL paths is lower for 30s switching time. We repeat this experiment for a 60s switching time but do not include the plot here as it is similar to 3.3(a) with larger intervals between domain switches. The other controller that we show in Figure 3.3(b) is the load balancer which checks the cumulative throughput on each of the VLANs whenever a flow arrives and then sends that flow out of the VLAN that has the least load while streaming the DASH video. Figure 3.5 shows a graph of the instantaneous throughput for each of the paths while running the Nowcast application as a standalone experiment using the load balancer controller. From this figure and Figure 3.3(b), we see that the throughput on Internet2-AL2S and ORNL goes to a higher peak within the first 100s and 400s respectively but after this time, all three paths share the load almost equally. Figure 3.4 shows the VLC DASH client performance for the timer-based controllers (both 30s and 60s switching intervals) and the load balancer. The controller that switches paths every 30s gives a higher, constant decision rate compared to the controller that switches every 60s but the best decision rate curve is obtained with the load balancer. On comparing figures 3.3(a), 3.3(b), and 3.5 it is evident that the paths are also better utilized by the load balancer controller as it gives a more distributed load and avoids the peaks that are seen in Figure 3.3(a).



(a)



(b)

Figure 3.3: Multi Domain VLC DASH Throughput

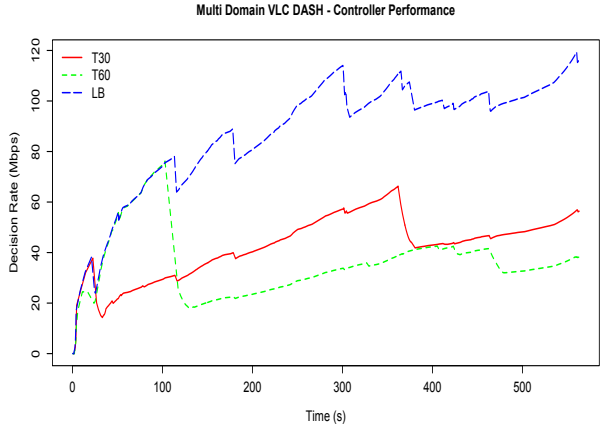


Figure 3.4: Multi Domain VLC DASH client

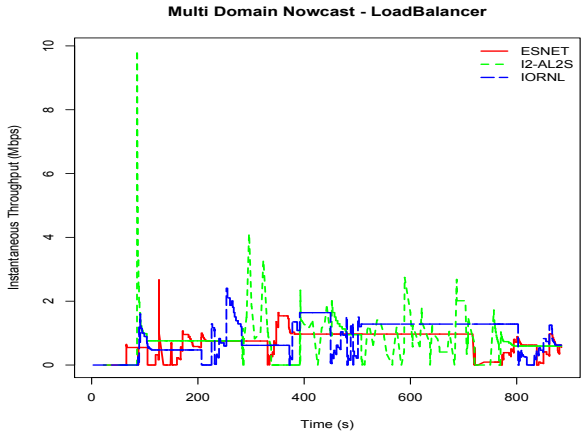
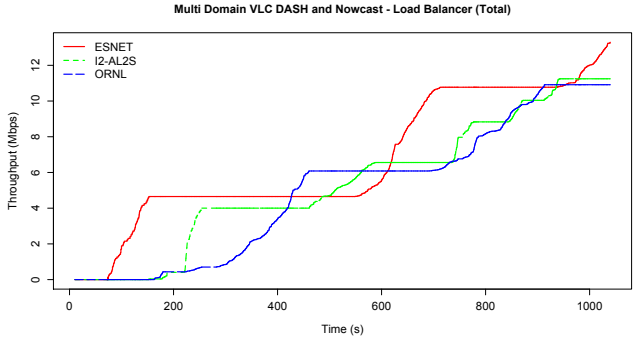
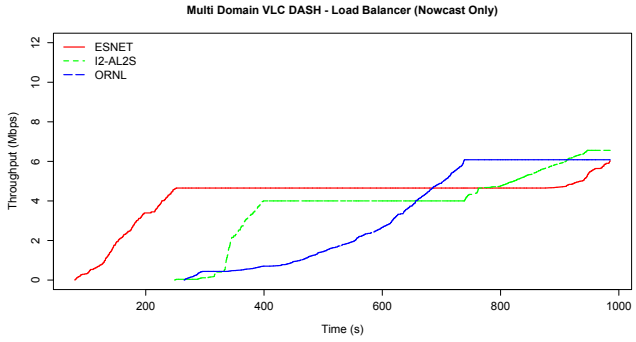


Figure 3.5: Multi Domain Nowcast



(a)



(b)

Figure 3.6: Multi Domain Nowcast and VLC DASH Throughput

Domain Type	Controller	Round-trip Time [ms]
Single Domain (ESNET)	non-SDN	99.96
Single Domain (ORNL)	non- SDN	136.40
Single Domain (I2-Al2S)	SDN	121.46
Multi Domain	Timer-based (60s)	75.00
Multi Domain	Timer-based (30s)	88.10
Multi Domain	Load-Balancer	28.31

Table 3.1: Average Round-trip time of ICMP packets between Northwestern and Atlanta

3.3.3 Multi Domain Multi Application Performance

Figure 3.6 illustrates two cumulative throughput graphs obtained while running the load balancer with both the Nowcast and VLC DASH applications. To analyze the performance for each application, in Figure 3.6(b), we plot the Nowcast flow statistics separately for the same time window in the experiment. At 400s, we start the VLC client application and the cumulative throughput after this point is greater for each path than seen in Figure 3.6(b). From Figure 3.6(a), we can clearly observe that load balancing gives a consistent network utilization even when multiple applications are run.

Table 3.1 shows the Round-trip Time (RTT) in milliseconds obtained for all cases described in this section. The ping utility with a packet size of 64 bytes was used to measure these values. The experiment consists of 1000 pings running in parallel from the processing box to the radars and from each of the radars to the processing box. The average value of these RTTs obtained with each type of domain and controller is shown here. As we can see above, using the multi-domain reactive controllers clearly reduces end-to-end latency between the Northwestern rack and the SoX rack. The best performance is obtained with the load-balancer algorithm with an average RTT of 28.31ms.

From our results here, we infer that SDXes have the capability to enhance the untapped potential of IXPs through intelligent traffic engineering. It is also interesting to note that the OpenFlow controller code for these experiments was easy to develop and test, thus, showing that SDXes can be easily integrated into applications. We have seen in other works such as [41] and [24] that video streaming applications can benefit significantly from traffic engineering in software-defined networks. In addition to allowing similar SDN capabilities, SDXes also allow dynamic resource allocation where the resources could include network, compute or storage. Although the results here are more a proof-of-concept than a real deployment, they provide a basis for virtualizing OpenFlow APIs for improving application delivery in content distribution systems. In the following sections, we explore such a design, present some examples of algorithms that use these virtual APIs and use a GENI testbed network to analyze the performance of a VLC DASH client that uses these APIs.

CHAPTER 4

SDX-CDN: AN ARCHITECTURE FOR CONTENT DELIVERY USING SDXes

In the previous section we examined the possible benefits of using SDXes to improve video delivery and performance. The proliferation of CDNs in the Internet ecosystem has given rise to interesting research on performance of CDNs and CDN peering [43] [48]. Since IXPs are an integral part of the Internet, we envision SDXes are an appropriate location to host on-demand content distribution network services. In this section, we describe an architectural design for such a system. Figure 4.1 shows the components of this system for a dynamic adaptive video streaming application where feedback is provided by a Software Defined Infrastructure (SDI).

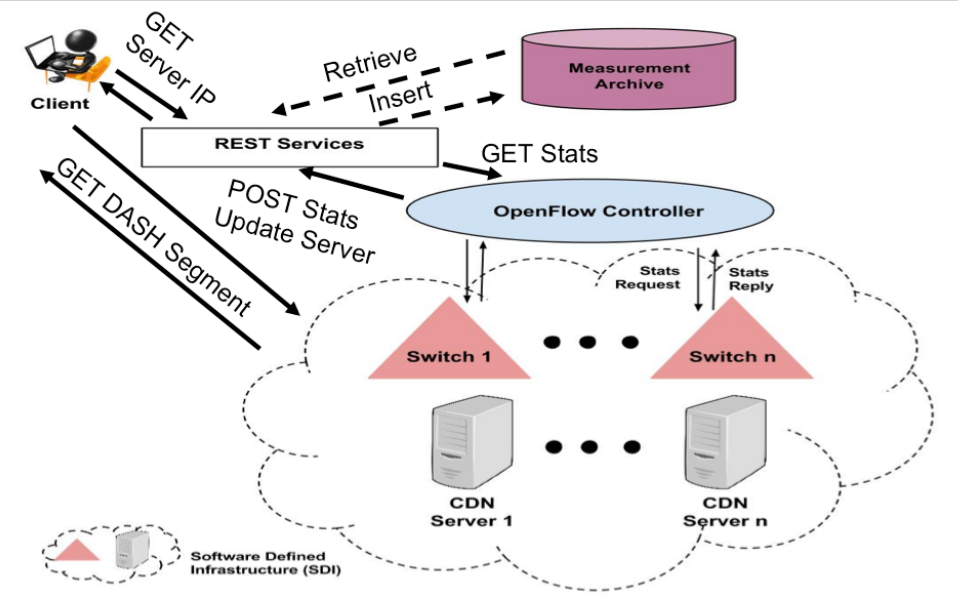


Figure 4.1: SDX-Application Service Architecture

4.1 Software Defined Infrastructure (SDI)

There are several definitions of SDI at [44] and [51]. Here we use the term SDI to refer to a network of software-defined switches (OpenFlow-enabled) particularly at Software Defined Exchanges that are co-located with storage and compute power. We define an architecture for SDIs to provide application-oriented services for Dynamic Adaptive Streaming over HTTP (DASH). Virtualized instances of SDXes can serve as CDNs because these instances are on-demand, elastic and inexpensive when compared to dedicated servers [18]. SDIs simplify third-party policy implementation and provide network administrators with the ability to provision, monitor and control virtual networks in an efficient manner. An example of a fully operational network that offers SDI capabilities is the GENI network [17], which is a deeply-programmable testbed that uses OpenFlow for routing in a network that consists of several different racks of varying compute and storage sizes.

4.2 Traffic Engineering

In this section we describe our traffic engineering approaches to improve application delivery performance in a CDN. OpenFlow [39], a proven technology for intelligent traffic engineering in production WANs [29] and enterprise data centers [58], is used to implement a feedback-based measurement and control system for a DASH video distribution system. A programmable OpenFlow controller probes switches in the SDI network for port statistics at custom-defined intervals according to the requirements of the application. The statistics are then collected in a distributed storage system described in Section 4.3 below. A spanning tree of the network topology is constructed at the controller with all possible Label Switched Paths (LSPs) between servers and clients. A custom-defined algorithm can be implemented at the controller to perform intelligent application-based routing. Examples of such algorithms are given in Chapter 5.

4.3 REST Interface

The measurement archival system is a distributed database provided by the open-source MongoDB software [3]. MongoDB implements security, quick insert and retrieval times for shallow searches. More importantly, it provides a distributed system for fast and easy replication thus, allowing redundancy. MongoDB also provides REST interface APIs for convenience [8]. This means information can be inserted and posted with a simple HTTP command without the need for additional software installation. The MongoDB interface can also be configured to set access rights and permissions for the archival data. Some examples of HTTP REST commands that can be run against our interface are:

Example 1:

```
curl -X GET http://<rest_interface_IP>:27080/opencdn/portmonitor/_find?
   criteria={"dpid":"26-e8-94-3a-a1-40","portno":2};batch_size=1
```

Code 4.1: REST API Example 1: HTTP GET

This command returns the latest entry for received bytes (RxBytes), transferred bytes (TxBytes), received packets (RxPackets) and transferred packets (TxPackets) for a switch with datapath id as *26-e8-94-3a-a1-40* and port number as *2*. In our current experiments, we have an Open API for this function to be used in the controller. Additionally, it can be used by an external sampling algorithm that performs post-processing of this data offline. An example of such a use case is described in Section 6.4.

Example 2:

```
curl -X GET http://<rest_interface_IP>:27080/opencdn/serv_bandwidth/
    batch_size=1
```

Code 4.2: REST API Example 2: HTTP GET

This command returns the server with the lowest bottleneck bandwidth to a given client from the set of servers in the network. This value is updated at intervals defined as the *Update Time* and directly affects how often the VLC client will switch servers during playback.

Example 3:

```
curl --data docs=[{"dpid":"26-e8-94-3a-a1-40"},{"portno":3}, {"RXpackets
    ": 1234}, {"RXbytes": 12340},{"TXpackets": 10}, {"TXbytes": 100}, {"
    date": datetime.utcnow()}
]' 'http://<rest_interface_IP>:27080/opencdn/portmonitor/_insert'
```

Code 4.3: REST API Example 3: HTTP POST

This command inserts the port statistics into the MongoDB database. We execute this command inside the *stats reply* event in the monitoring section of our controller. This command can be further extended to insert any and all statistics provided by OpenFlow.

4.4 Application

We use a VLC client and an Apache2 server to provide video distribution services using the SDI defined in 4.1. The Apache2 servers are installed in a virtualized cloud instance at the SDX. The VLC client is an open-source video player that comes with inbuilt plugins for playing DASH videos. The adaptation algorithm implemented in the VLC DASH plugin decides the bitrate at the which to request segments in a video. By default, it uses a combination of buffer fill percentage and current segment download rate to decide the quality at which the following segment is requested. In this implementation, the client chooses to request all segments of a video from a single server. We modify this feature such that the client now makes a REST call to the SDI which returns a CDN server IP address to the client to request the next segment from. This IP address will have been inserted into a table of the MongoDB archive by the OpenFlow controller based on the algorithm as described in Section 4.2 above.

CHAPTER 5

SDX-CDN: AN EXPERIMENTAL EVALUATION

In this section, we describe our experiment scenarios that test the basic functionality of our system and then evaluate more sophisticated approaches based on real cross traffic patterns seen in the Internet.

5.1 Testbed Network

Figure 5.1 shows our experimental setup using the GENI testbed [17]. We use Xen Virtual Machines (VM) located in the Utah Downtown Data Centre (UtahDDC). Although these VMs are all geographically colocated, they are in placed in different racks in order to avoid back-plane bandwidth effects in our measurements. The prototype setup consists of Xen VMs with Open Virtual Switch (OVS) software, which uses OpenFlow 1.0 and emulates an SDX node. Additionally, the nodes labelled server$X$ run Apache2 servers preloaded with the Big Buck Bunny dataset, client1 and client2 are used for cross traffic generation while client3 is pre-installed with the VLC DASH client software.

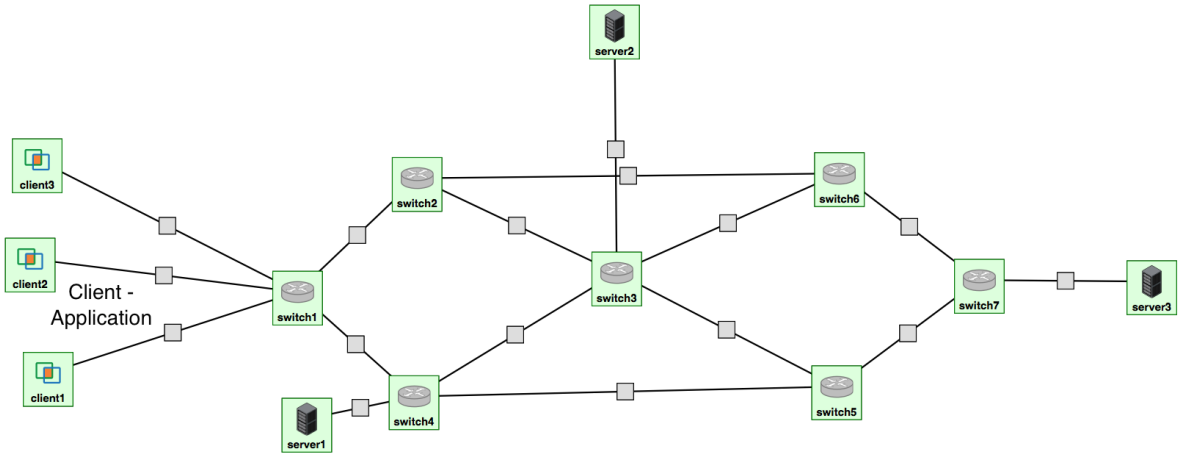


Figure 5.1: GENI Testbed Topology

5.2 OpenFlow Controller

The controller is programmed using POX open APIs and is based on previous work by Adrichem et al [57]. Although we use a Trema controller for our previous work described in Section 3, we replace this with POX here as it provides a more comprehensive set of OpenFlow APIs and better development support. However, the REST API framework is independent of the type of controller API used as the interaction occurs through HTTP GET and POST requests that can be executed in most controller applications.

5.2.1 Monitoring

The monitoring module is responsible for gathering statistics from switches and inserting these stats into the MongoDB archive. Currently, we obtain and process the instantaneous port statistics to reflect real-time bottleneck bandwidth for each link along a given path. The port statistics provide the received (Rx) bytes and transferred bytes (Tx) for each port on a switch and is updated every second.

5.2.2 Forwarding

The forwarding module is responsible for executing a feedback algorithm to perform reactive control of DASH video traffic. We use the same module to control our cross traffic flows as well. An example of POX controller code that executes such a forwarding algorithm for VLC client flows is given here:

```
for i in range len(num_of_servers)
      for j in range len(num_of_paths_to_server):
       min_bw=0
       for k in range (num_of_switches_in_path):
          result=(HTTP GET against REST interface with criteria as
              switch(k) and outgoing port from path j )

          min_bw = max(min_bw, result[TXbytes])
       if best_bw > min_bw:
        best_bw = min_bw
        best_serv=i

       Update traffic matrix, REST interface
```

Code 5.1: REST API Use: Example

The algorithm above contains an implementation of Equations 5.1 and 5.2 described later in this Section 5.5.3 and updates the traffic matrix in our controller. This pseudo-code iterates through each path in the server and queries the REST interface for the amount of transferred bytes on the outgoing port of every switch on that path. It then computes the maximum transferred bytes for every path given by *min_bw*. The best server is then determined as the server with the minimum *min_bw* and is denoted by *best_bw*. This information is then updated into the traffic matrix and the REST interface which then inserts it into the database. The cross traffic design is described in the following sections.

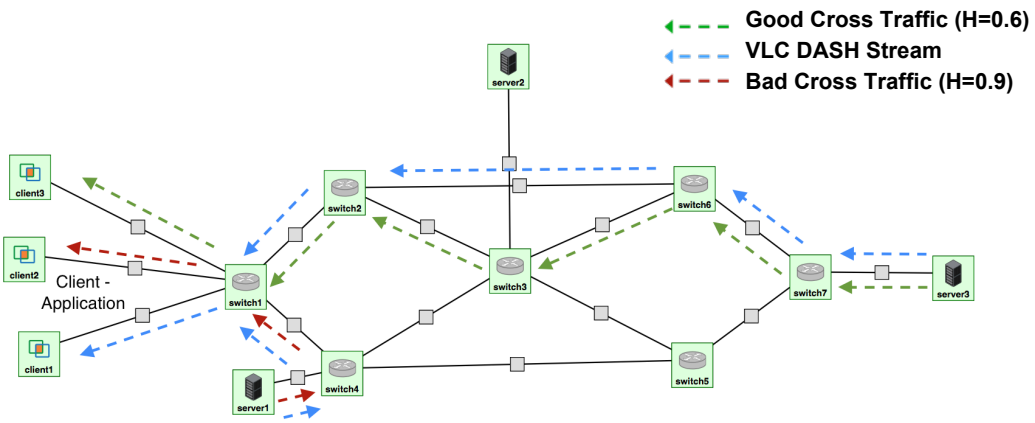


Figure 5.2: GENI Testbed Topology with Traffic Flows

5.3 Cross Traffic

Figure 5.2 shows the paths taken by cross traffic and application flows to each of the servers. The red and green dashed lines represent cross traffic while the blue line represents VLC DASH application flows. We choose these paths such that the video stream is sufficiently throttled by the cross traffic. We also choose two paths of distinct round trip times and hop counts but we expect these factors to have a negligible effect on video stream quality, which is mostly affected by bottleneck bandwidth on the path. It has been shown in previous research [14] that video freezes are a result of small buffer sizes that occur due to large delays in the network. Since our testbed setup is fairly small, we are not concerned with video rebuffering in this analysis.

5.3.1 UDP Iperf

To show the effects of throttling on the VLC DASH application, we use UDP Iperf to generate packet bursts of increasing constant bit-rate (CBR). CBR traffic helps us to evaluate the performance of our system as the duration and magnitude of the UDP bursts determine the available bandwidth on a chosen path at any given time. Analyzing these results as time-series data demonstrates a proof-of-concept for our system. Thus, we study the effectiveness of switching servers during video streaming to provide better quality to the client.

5.3.2 Self-Similar

Several works have shown that Internet traffic exhibits Long-Range Dependence (LRD). Leland et al. studied Internet traffic patterns using traces of long duration, i.e, greater than 24 hours as early as 1994 and observed that when a small period of this trace is chosen and magnified it exhibits self-similar patterns similar to those seen in fractals [37]. The chosen period is magnified into sub periods and each of these sub periods consists of large bursts separated by smaller bursts of traffic. This "burstiness" can be quantified by a parameter called the Hurst factor, which ranges anywhere between 0.5 and 1. A Hurst factor of 0.5 indicates negligible self-similarity and a Hurst factor of 0.9 indicates a high degree of self-similarity where the normalized shape of the sub-periods exhibit high fractal-like behaviour. We use this cross traffic to verify the usefulness of switching servers at different points in DASH video playback. Figure 5.2 shows a green line that represents an LRD flow with a Hurst factor of $H=0.6$ which we call "good traffic" and the red line represents an LRD flow with a Hurst factor of $H=0.9$ which we call "bad traffic". This traffic exhibits self-similar behaviour with an average rate of 75Mbps over a period of 800s.

5.4 Application

5.4.1 Content Distribution Network for Video Distribution

The CDN network consists of three servers hosted on Xen Virtual Machines (VM) at three different racks in the GENI UtahDDC aggregate. We use the Big Buck Bunny dataset, which is provided by ITEC [34] and lasts for about ten minutes. An Apache2 server is installed in each of these VMs that have a single core, 1 GB RAM and 6 GB disk space.

5.4.2 VLC - DASH

We use the VLC DASH application to evaluate our system performance for adaptive streaming applications. Although the VLC is a full video player, we use the headless VLC in our setup because our testbed consists of VMs that are not equipped with a GUI. This VLC application is altered to do the following:

1) Allow collection of measurements through the definition of measurement points using Orbit Measurement Library (OML) software [52] and

2) Create a HTTP GET request to the REST Interface that returns the IP address to request DASH segments from.

5.5 Experiments

5.5.1 Throttling Effect

This experiment is similar to the system employed by NetFlix and Hulu as described in [13] and [12] respectively. In the systems described in [13] and [12], a client is forced to stay with a single server for the entire duration of video playback until the bitrate quality drops to the lowest value. It is evident from previous research that this is undesirable to a user as it degrades the QoE [61]. In this experiment, we try to replicate similar client behaviour in the VLC player. We initiate cross traffic UDP Iperf flows from server3 and client2 and server1 and client1 as shown in Figure 5.1 with with parallel flows of 40MBps of 30s each between each of them.

5.5.2 Proof-of-Concept

In this experiment, we use UDP Iperf to show the behaviour of our system under throttling effects. For traffic initiating from server3, we begin with 5 parallel flows and decrement this number every 30s and for traffic flowing from server1 we increment the number of parallel flows from 1 to 5 every 30s. We design this experiment as a proof-of-concept to observe how the VLC DASH client is affected by requesting subsequent segments from multiple servers.

5.5.3 Instantaneous Available Bandwidth

A traffic matrix in the controller contains information about all possible paths for each server in the network. The bottleneck bandwidth for each path is updated at a custom-defined time interval.

$$MinPath(i,j)(t) = Min(Max(Tx\_Bytes\,on\,outgoing\,link))$$
$$\forall\ i \in \{paths\ \ to\ \ server\,j\}$$
$$j \in \{list\,of\ servers\}\,at\ time = t \tag{5.1}$$

$$BestServer(t) = Min(MinPath(i,j)),$$

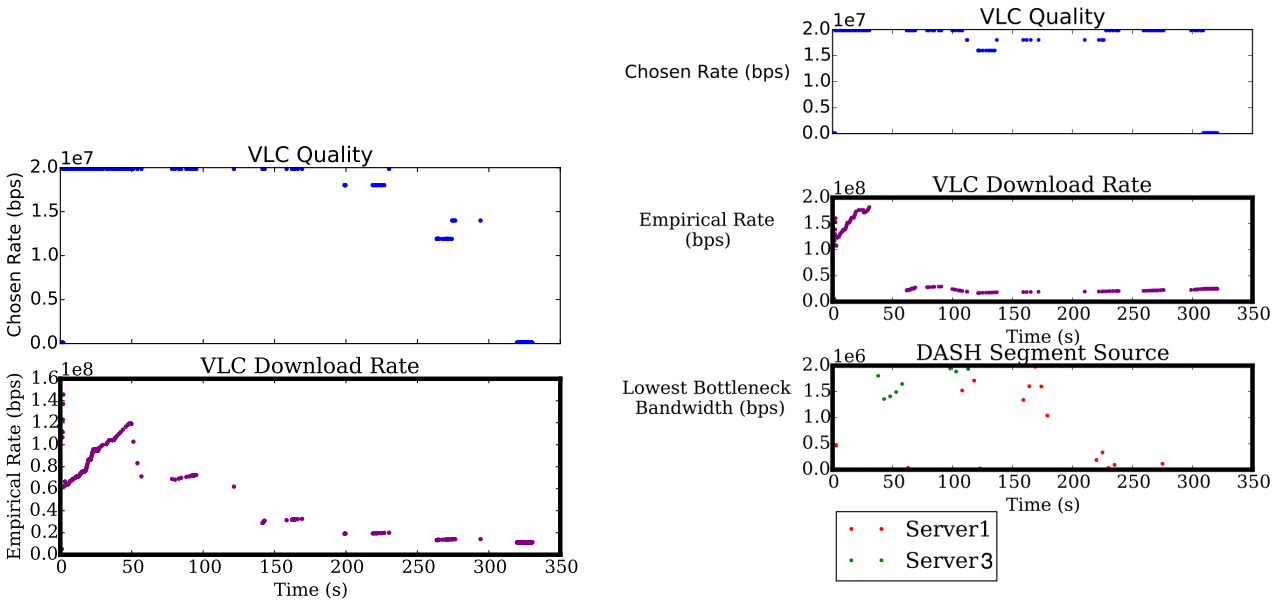
$$where\, j \in \{list\, of\, servers\}\, at\, time = t \tag{5.2}$$

The bottleneck bandwidth of a path between the client and a server is calculated as the maximum transferred bytes on every outgoing port along that path. The $MinPath(i,j)(t)$ at time=t is then computed as the path with the lowest bottleneck bandwidth as shown in Equation (5.1). The controller then updates the REST Interface with the best server ($BestServer(t)$) at time=t by selecting the server which has the least bottleneck bandwidth of all its available paths as shown in Equation (5.2). When a segment request packet from the client reaches the controller, it chooses the least congested path to the server as seen in the traffic matrix. Code 5.1 provides the pseudo-code for this algorithm. The network information is thus, abstracted from the client. This experiment is repeated with the instantaneous bandwidth updated to the traffic matrix every 2s and 5s.

CHAPTER 6

SDX-CDN: RESULTS AND ANALYSIS

In this section, we analyze the experiments described in Chapter 5. Towards the end of this chapter, we also present a practical use case that integrates a sampling and network path generating module with our system.

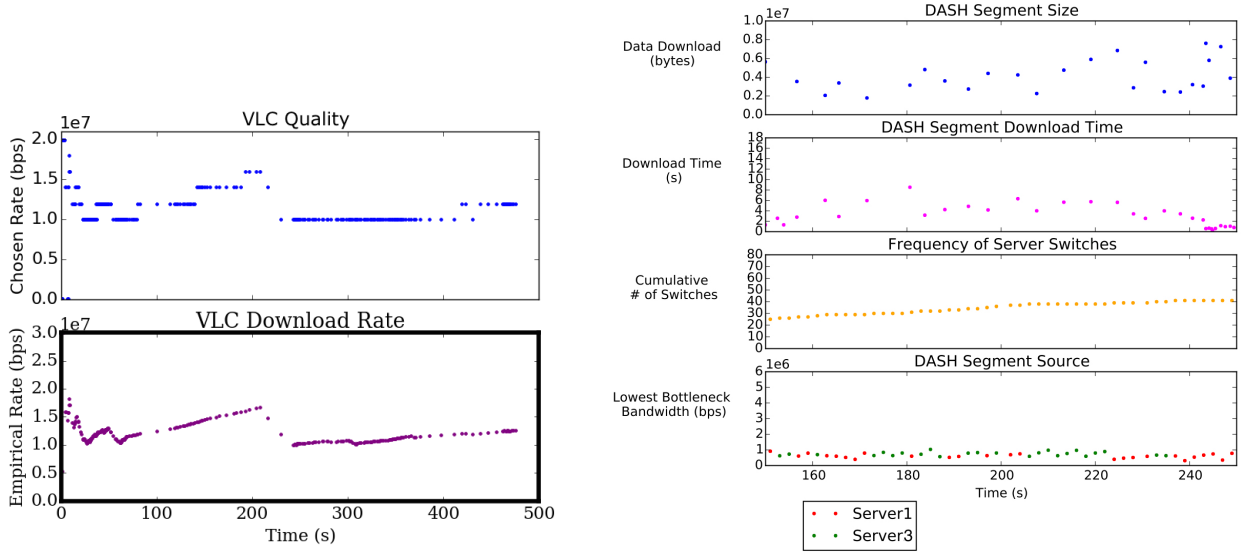


(a) (b)

Figure 6.1: Throttle Bandwidth : (a) Single Server (b) Two Servers

6.1 Throttling Effects

In Figure 6.1(a), we observe how the bitrate quality of the VLC DASH client is affected by increasing the number of parallel UDP Iperf flows to server1. Streaming from server3 exhibits a similar behaviour and is thus, not shown here. Based on observations by Adhikari et al. [13], Netflix would allow a similar client to switch to a different server only at about 300s when the quality has dropped to the lowest bitrate. However, the client has already seen a significant degradation in video quality as early as 250s. This method is adopted to minimize the TCP slow start problem that occurs when the client terminates an existing connection and makes a new one to a potentially better server. As studied by researchers in [61] and [32], quality drops and rebuffering during watching of the video leads to a high level of dissatisfaction for the user and could lead to video abandonment if it occurs frequently. It is, therefore, not desirable to stream from a single server after a DASH client experiences degradation in bitrate quality. However, frequent switching of servers during video playback leads to oscillations in bottleneck bandwidth and thus, oscillation in video quality bitrates which is undesirable. In subsequent results, we see that there is much to gain from switching servers during DASH video playback if the *Update Time* is not too small. We define the *Update Time* as the time between server updates in the REST interface and thus, this *Update Time* directly determines the frequency of server switches in

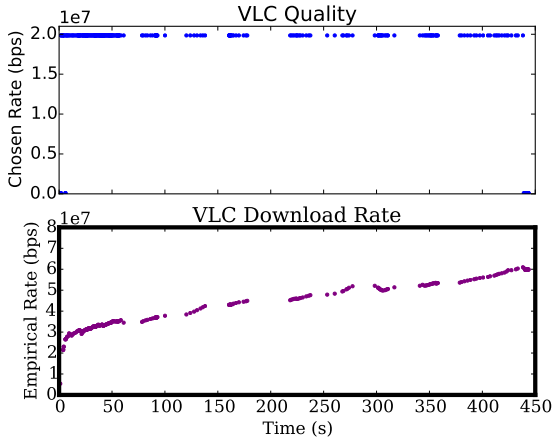


(a) (b)

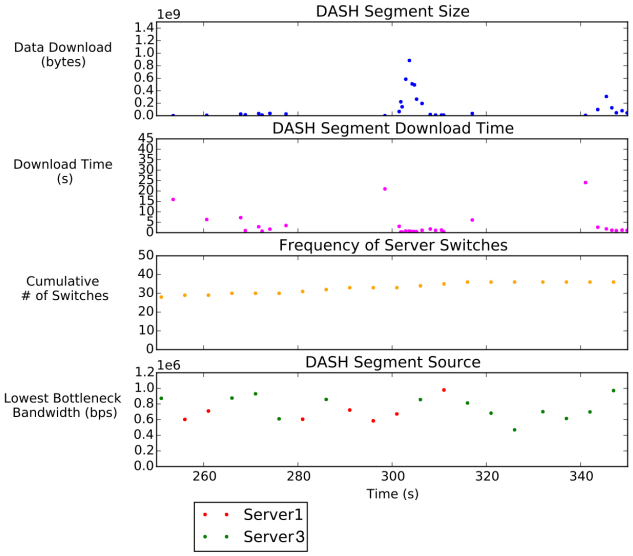
Figure 6.2: Multiple Servers - 2s Update Time

6.2 Proof-of-Concept

Figure 6.1(b) shows the behaviour of our system when the paths to the servers are throttled at different times during the playout of the video. Initially, the number of parallel flows from server1 are higher and the client, therefore, chooses to stream 100s of video from server3. However, after this time, server3 has increasing parallel flows, and the system recomputes the traffic matrix after which it updates the database with the best server changed to server1. The traffic matrix is updated every 5s in this case. Since this experiment was run with CBR cross traffic that changed every 30s, we do not believe it is interesting to use varying update times here. The VLC DASH client requests the first segment at an arbitrary point in time after the start of the Iperf flows. In Figure 6.1(b), we see the bitrate quality drop just before 150s. This is the mid-point at which both servers have equal number of parallel flows before server1 reduces the number of Iperf flows and server3 continues to increase the number of flows. After verifying the general functionality of our system, we changed the cross traffic to a probabilistic LRD traffic as described in Section 5.3.2 and observe the effects of varying traffic matrix update times (*Update Time*).



(a)



(b)

Figure 6.3: Multiple Servers - 5s Update Time

6.3 Instantaneous Available Bandwidth

The results for varying traffic matrix update times and their effects on VLC DASH bitrate quality is shown in Figures 6.2 and 6.3. In Figure 6.2(a), the VLC DASH quality oscillates between two levels until 90s and drops significantly at 220s. The download rate which the VLC uses to estimate the best quality to request next also varies with a similar shape. This can be compared with the results of updating the traffic matrix every 5s, where the client does not see any drop in quality but instead, it sees a consistently high quality bandwidth. Figures 6.2(b) and 6.3(b) contain sub periods from 6.2(a) and 6.3(a) respectively, to illustrate the effects of switching servers too often. From Figure 6.2(a) that shows a 100s sub period, we see frequent server switches with a cumulative server switch of 16 in this sub period alone. The aggregate number of server switches is as high as 80 for the duration of the video. We also note that segment download times are large even for smaller segments as a results of switching servers. This results in a skewed estimate of the download rate and thus, lower quality for the VLC client. However, in Figure 6.3(a) we see that the server switches only 8 times in 100s and only 50 times overall. The segment download

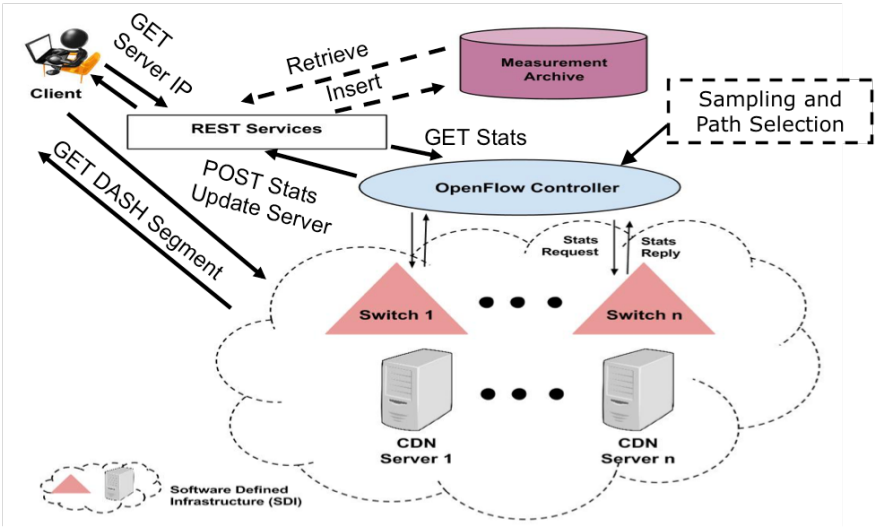


Figure 6.4: SDX-Application Service Architecture - Module Plugin

6.4 Use Case [1]

[2]This section describes a real use case for our system that contains a plug-and-play sampling and path selection algorithm. Studies by several researchers presented in Section 2.3 have shown that separation of the control plane from the data plane in SDN opens up a plethora of possibilities for dynamic control of video streaming applications. Moving this control plane computation to a processing environment with highly powerful resources can aid in improving QoS in SDN networks. In this use case, we present a novel approach for reducing the overhead due to flow monitoring and show how a fine-grained characterization of a flow auto-correlation structure can improve VLC DASH client performance. Such sophisticated path computation algorithms can be seamlessly integrated into our system to provide network administrators with fine-grained control over the CDN as shown in Figure 6.4. The sampling and path selection algorithm is integrated as a Python module into our control and monitoring framework described in Section 5.2. The testbed setup is similar to the one described in Section 5. The following subsections describe this module in more detail.

[1]This work is to appear at Infocomm 2016 and was done in collaboration with Zdravko Bozakov (Leibniz Universität Hannover) and Amr Rizk (University of Massachusetts, Amherst)

[2] We give here an overarching description of the methodology. All detailed technical derivations have been omitted as it is not within to scope of this document.

6.4.1 Random Sampling

In order to obtain a sample space that contains sub periods of different granularities with minimal overhead, the sampling module used here is based on a random coin flip that decides with a probability of P=0.5 whether to query statistics from each switch in the network. A sliding window sampling method provides an auto covariance traffic matrix which is used to generate multiple sample paths for buffer overflow estimation. The buffer overflow estimate is the probability with which the queues in the OVS switches in our network, as described in Section 5.1, are flooded with more packets than they can process at a given point in time. A covariance matrix is used to generate multiple possible paths instead of using a single one.

6.4.2 Path Generator

The Cholesky decomposition [26] method is used to generate independent sample paths from the auto covariance matrix described above. These paths could then be simulated to analyze metrics such as delay distributions and buffer overflow probability estimations. This method is based on the assumption that Internet traffic exhibits a Gaussian distribution. In a testbed VM, this algorithm takes several seconds to compute and is ideally implemented in an external hardware module. However, this decision is dependent on the path computation algorithm and is determined by the network administrator independent of the design parameters of our system. For a non-Gaussian distribution, we do not implement the analysis here but it can be found in [1].

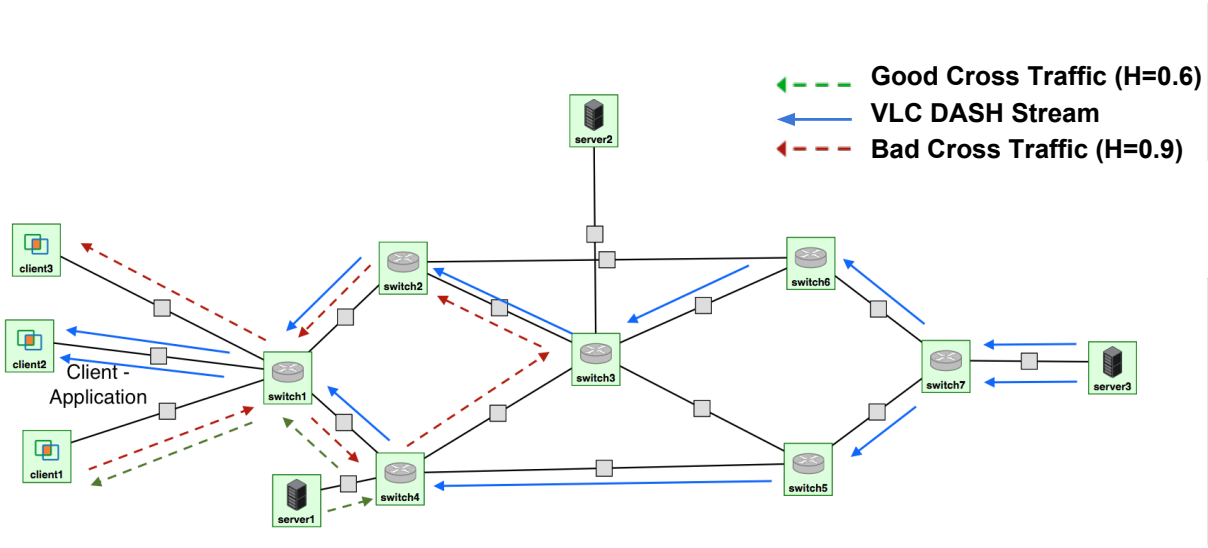


Figure 6.5: GENI Testbed Topology with Traffic Flows - Sample Path Generation

6.4.3 VLC DASH

We use two paths as shown in Figure 6.5 to stream the VLC video from server3. The cross traffic is allowed to flow on two separate paths similar to the experiment described in Section 5.3.2 and seen in Figure 6.5 as the red and green dashed lines. The average utilization is 0.75 and the Hurst factor used to indicate the "burstiness" of the traffic is set to *H=0.6* and *H=0.9* for the two paths respectively.

The graph in Figure 6.6 clearly shows that the OpenFlow controller with the path generator plugin differentiates between the different levels of "burstiness" to select a better path for the VLC DASH stream and thus, provide higher quality bitrates.

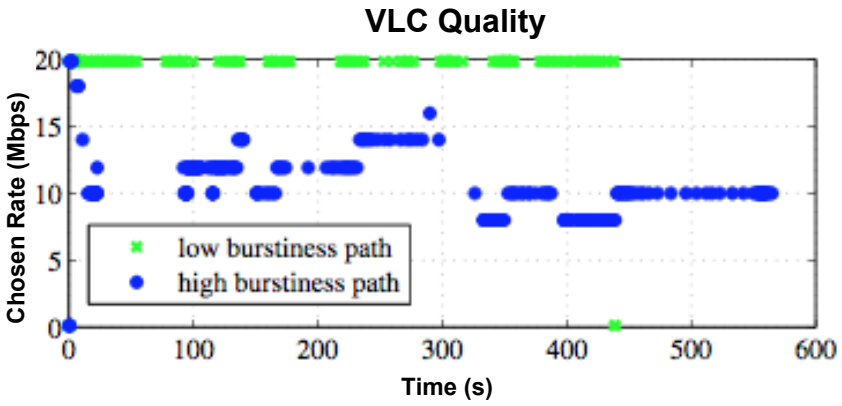


Figure 6.6: Single Server - Multiple Paths

In this section, we present several interesting results, consisting of initial working examples, a smart server-selection algorithm and finally an actual use case of our system for sophisticated monitoring and path computation algorithms. It is evident from these results that the interaction of a dynamic adaptive video streaming with a network monitoring and control framework offers significant improvements in bitrate qualities. It is also noteworthy that OpenFlow controller APIs provided by tools such as POX [7] and Trema [9] make it easy to build and develop such a system within a short span of time. Additionally, the work in 6.4 above shows a real-world example that integrates seamlessly with our framework and demonstrates a typical use case for our system to improve adaptive video delivery by switching traffic flow paths in an SDN network.

CHAPTER 7

CONCLUSION

Our work shows that Software Defined Exchanges when thought of as an IXP co-located with compute and storage resources can function as a third party to implement policies between service and content providers. Application aware routing using Northbound APIs in the OpenFlow implementation is currently being evaluated by few researchers but their work has also shown that application friendly features of SDN such as traffic prioritization, VLAN provisioning, packet reformatting, policy definitions and so on can be exploited to improve Quality of Service (QoS) for the user. We have seen in Section 3 that co-location of compute and storage resources can have significant advantages for applications such as weather data processing, which needs to be done in a timely and accurate manner. In later sections, we describe a prototype architecture for video streaming content distribution using SDXes. Our results demonstrate a virtual API framework that seamlessly integrates with an OpenFlow controller and an adaptive streaming application to provide improved video quality bitrates. We have also seen through an interesting use case, how easily our framework can interface with a plugin sampling-and-path selection algorithm to perform intelligent flow processing and provide a least-congested path for DASH traffic. While we do not provide conclusive evidence that SDXes must be implemented in the Internet architecture as we know it today, we are convinced that this is a possibility well worth investigating.

7.1 Future Work

An open question that currently exists with our system as with several SDN systems is that of scalability. While this design is meant to be an initial prototype, our next goal is to make this scalable and examine the performance for a larger network of a hundred nodes or more. We are curious about the behaviour of this system for multiple clients and expect that SDN multicasting [25] approaches could easily integrate with and contribute significantly to improving the scalability of such a system. We are also looking into designing a controller for the weather application mentioned in our earlier work presented in Chapter 3 so that we may be able to understand and improve the performance of high priority applications such as weather. This can then be contrasted and compared with our video application performance results to extend this design to include a plugin module that accepts QoS metrics and returns a customized traffic engineering model as an output to be directly executed in an application specific content delivery network.

BIBLIOGRAPHY

[1] Akamai AT&T. http://www.akamai.com/html/about/press/releases/2012/press_120612.html.

[2] Internet2-Advanced layer 2 Services(AL2S). http://noc.net.internet2.edu/i2network/advanced-layer-2-service/maps-documentation/al2s-topology.html.

[3] MongoDB. https://www.mongodb.org/.

[4] NetFlix OpenConnect. https://www.netflix.com/openconnect.

[5] NFV. http://portal.etsi.org/nfv/nfv_white_paper.pdf.

[6] Peering DB. https://www.peeringdb.com.

[7] POX. http://www.noxrepo.org/pox/about-pox/.

[8] Sleepy Mongoose (MongoDB). http://www.kchodorow.com/blog/2010/02/22/sleepy-mongoose-a-mongodb-rest-interface/.

[9] Trema. http://trema.github.io/trema/.

[10] NLANR/DAST : Iperf - the TCP/UDP bandwidth measurement tool. http://dast.nlanr.net/Projects/Iperf/, Accessed 2007.

[11] Local data manager (ldm). http://www.unidata.ucar.edu/software/ldm/, Accessed 2014.

[12] Adhikari, Vijay Kumar, Guo, Yang, Hao, Fang, Hilt, Volker, and Zhang, Zhi-Li. A tale of three cdns: An active measurement study of hulu and its cdns. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on* (2012), IEEE, pp. 7–12.

[13] Adhikari, Vijay Kumar, Guo, Yang, Hao, Fang, Varvello, Matteo, Hilt, Volker, Steiner, Moritz, and Zhang, Zhi-Li. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *INFOCOM, 2012 Proceedings IEEE* (2012), IEEE, pp. 1620–1628.

[14] Akhshabi, Saamer, Begen, Ali C, and Dovrolis, Constantine. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of the second annual ACM conference on Multimedia systems* (2011), ACM, pp. 157–168.

[15] Bari, M.F., Chowdhury, S.R., Ahmed, R., and Boutaba, R. "policycop: An autonomic qos policy enforcement framework for software defined networks," future networks and services (sdn4fns), 2013 ieee sdn for , vol., no., pp.1,7, 11-13 nov. 2013.

[16] Bennesby, R., Mota, E., Fonseca, P., and Passito, A. Innovating on interdomain routing with an inter-sdn component. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on* (May 2014), pp. 131–138.

[17] Berman, Mark, Chase, Jeffrey S., Landweber, Lawrence, Nakao, Akihiro, Ott, Max, Raychaudhuri, Dipankar, Ricci, Robert, and Seskar, Ivan. Geni: A federated testbed for innovative network experiments. *Computer Networks 61*, 0 (2014), 5 – 23. Special issue on Future Internet Testbeds Part I.

[18] Buyya, Rajkumar, Yeo, Chee Shin, Venugopal, Srikumar, Broberg, James, and Brandic, Ivona. Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems 25*, 6 (2009), 599 – 616.

[19] Chatzis, Nikolaos, Smaragdakis, Georgios, Feldmann, Anja, and Willinger, Walter. There is more to ixps than meets the eye. *ACM SIGCOMM Computer Communication Review 43*, 5 (2013), 19–28.

[20] Chetty, Marshini, and Feamster, Nick. Refactoring network infrastructure to improve manageability: A case study of home networking. *SIGCOMM Comput. Commun. Rev. 42*, 3 (June 2012), 54–61.

[21] Chowdhury, Shihabur Rahman, Bari, Md Faizul, Ahmed, Reaz, and Boutaba, Raouf. Payless: A low cost netowrk monitoring framework for software defined networks. In *14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014)(To appear)* (2014).

[22] Feamster, Nick, Rexford, Jennifer, and Zegura, Ellen. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review 44*, 2 (2014), 87–98.

[23] Frank, Benjamin, Poese, Ingmar, Lin, Yin, Smaragdakis, Georgios, Feldmann, Anja, Maggs, Bruce, Rake, Jannis, Uhlig, Steve, and Weber, Rick. Pushing cdn-isp collaboration to the limit. *ACM SIGCOMM Computer Communication Review 43*, 3 (2013), 34–44.

[24] Ganjam, Aditya, Siddiqui, Faisal, Zhan, Jibin, Liu, Xi, Stoica, Ion, Jiang, Junchen, Sekar, Vyas, and Zhang, Hui. C3: internet-scale control plane for video quality optimization. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015* (2015), pp. 131–144.

[25] Georgopoulos, P., Broadbent, M., Plattner, B., and Race, N. Cache as a service: Leveraging sdn to efficiently and transparently support video-on-demand on the last mile. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on* (Aug 2014), pp. 1–9.

[26] Golub, Gene H, and Van Loan, Charles F. *Matrix computations*, vol. 3. JHU Press, 2012.

[27] Gupta, Arpit, Vanbever, Laurent, Shahbaz, Muhammad, Donovan, Sean P., Schlinker, Brandon, Feamster, Nick, Rexford, Jennifer, Shenker, Scott, Clark, Russell J., and Katz-Bassett, Ethan. SDX: a software defined internet exchange. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014* (2014), pp. 551–562.

[28] Huang, Te-Yuan, Johari, Ramesh, McKeown, Nick, Trunnell, Matthew, and Watson, Mark. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM* (2014), ACM, pp. 187–198.

[29] Jain, Sushant, Kumar, Alok, Mandal, Subhasree, Ong, Joon, Poutievski, Leon, Singh, Arjun, Venkata, Subbaiah, Wanderer, Jim, Zhou, Junlan, Zhu, Min, Zolla, Jon, Hölzle, Urs, Stuart, Stephen, and Vahdat, Amin. B4: Experience with a globally-deployed software defined wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (New York, NY, USA, 2013), SIGCOMM '13, ACM, pp. 3–14.

[30] Jarschel, Michael, Wamser, Florian, Hohn, Thomas, Zinner, Thomas, and Tran-Gia, Phuoc. Sdn-based application-aware networking on the example of youtube video streaming. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on* (2013), IEEE, pp. 87–92.

[31] Koponen, Teemu, Amidon, Keith, Balland, Peter, Casado, Martín, Chanda, Anupam, Fulton, Bryan, Ganichev, Igor, Gross, Jesse, Gude, Natasha, Ingram, Paul, et al. Network virtualization in multi-tenant datacenters. In *Networked Systems Design and Implementation* (2014).

[32] Krishnan, S Shunmuga, and Sitaraman, Ramesh K. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *Networking, IEEE/ACM Transactions on 21*, 6 (2013), 2001–2014.

[33] Lantz, Bob, Heller, Brandon, and McKeown, Nick. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (New York, NY, USA, 2010), Hotnets-IX, ACM, pp. 19:1–19:6.

[34] Lederer, Stefan, Müller, Christopher, and Timmerer, Christian. Dynamic adaptive streaming over http dataset. In *Proceedings of the 3rd Multimedia Systems Conference* (2012), ACM, pp. 89–94.

[35] Lee, Jeongkeun, Uddin, Mostafa, Tourrilhes, Jean, Sen, Souvik, Banerjee, Sujata, Arndt, Manfred, Kim, Kyu-Han, and Nadeem, Tamer. mesdn: Mobile extension of sdn. In *Proceedings of the Fifth International Workshop on Mobile Cloud Computing & Services* (New York, NY, USA, 2014), MCS '14, ACM, pp. 7–14.

[36] Mambretti, Joe, DeFanti, Thomas A., and Brown, Maxine D. Starlight: Next-generation communication services, exchanges, and global facilities. *Advances in Computers 80* (2010), 191–207.

[37] Mandelbrot, Benoit B. *The fractal geometry of nature*, vol. 173. Macmillan, 1983.

[38] Mauthe, Andreas, and Plagemann, Thomas. Content networking: research challenges of future content distribution. *China communications* (2006), 91.

[39] McKeown, Nick, Anderson, Tom, Balakrishnan, Hari, Parulkar, Guru, Peterson, Larry, Rexford, Jennifer, Shenker, Scott, and Turner, Jonathan. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (Mar. 2008), 69–74.

[40] Monsanto, Christopher, Reich, Joshua, Foster, Nate, Rexford, Jennifer, and Walker, David. Composing Software-Defined Networks. In *USENIX NSDI* (2013).

[41] Mukerjee, Matthew K., Naylor, David, Jiang, Junchen, Han, Dongsu, Seshan, Srinivasan, and Zhang, Hui. Practical, real-time centralized control for cdn-based live video delivery. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, ACM, pp. 311–324.

[42] Müller, Christopher, and Timmerer, Christian. A vlc media player plugin enabling dynamic adaptive streaming over http. In *Proceedings of the 19th ACM international conference on Multimedia* (2011), ACM, pp. 723–726.

[43] Pathan, Al-Mukaddim Khan, Broberg, James Andrew, Bubendorfer, Kris, Kim, Kyong Hoon, and Buyya, Rajkumar. An architecture for virtual organization (vo)-based effective peering of content delivery networks. In *Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks* (2007), ACM, pp. 29–38.

[44] Raghavan, Barath, Casado, Martín, Koponen, Teemu, Ratnasamy, Sylvia, Ghodsi, Ali, and Shenker, Scott. Software-defined internet architecture: Decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2012), HotNets-XI, ACM, pp. 43–48.

[45] Rakotoarivelo, Thierry, Jourjon, Guillaume, Mehani, Olivier, Ott, Maximilian, and Zink, Mike. Repeatable experiments with labwiki. *CoRR abs/1410.1681* (2014).

[46] Rakotoarivelo, Thierry, Ott, Max, Jourjon, Guillaume, and Seskar, Ivan. Omf: A control and management framework for networking testbeds. *ACM Operating Systems Review (OSR) 43*, 4 (January 2010), 54–59.

[47] Rexford, Jennifer, Wang, Jia, Xiao, Zhen, and Zhang, Yin. Bgp routing stability of popular destinations. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2002), ACM, pp. 197–202.

[48] Rodrigues, M., Fernandes, S., Kelner, J., and Sadok, D. An analytical view of multiple cdns collaboration. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on* (May 2014), pp. 25–32.

[49] Rothenberg, Christian Esteve, Nascimento, Marcelo Ribeiro, Salvador, Marcos Rogerio, Corrêa, Carlos Nilton Araujo, Cunha de Lucena, Sidney, and Raszuk, Robert. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 13–18.

[50] Salsano, Stefano, Ventre, Pier Luigi, Prete, Luca, Siracusano, Giuseppe, and Gerola, Matteo. Oshi - open source hybrid ip/sdn networking (and its emulation on mininet and on distributed sdn testbeds). *CoRR abs/1404.4806* (2014).

[51] Simeonidou, D., Nejabati, R., and Channegowda, M.P. Software defined optical networks technology and infrastructure: Enabling software-defined optical network operations. In *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013* (March 2013), pp. 1–3.

[52] Singh, Manpreet, Ott, Maximilian, Seskar, Ivan, and Kama, Pandurang. ORBIT measurements framework and library (OML): Motivations, design, implementation, and features. In *TridentCom 2005, 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities* (Piscataway, NJ, USA, Feb. 2005), Javier Aracil, Shivkumar Kalyanaraman, and Kenichi Mase, Eds., IEEE Computer Society, pp. 146–152.

[53] Stockhammer, Thomas. Dynamic adaptive streaming over http–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems* (2011), ACM, pp. 133–144.

[54] Stringer, Jonathan, Pemberton, Dean, Fu, Qiang, Lorier, Christopher, Nelson, Richard, Bailey, Josh, Correa, Carlos N.A., and Rothenberg, Christian Esteve. Cardigan: Sdn distributed routing fabric going live at an internet exchange. In *Computers and Communication (ISCC), 2014 IEEE Symposium on* (June 2014), pp. 1–7.

[55] Suresh, Lalith, Schulz-Zander, Julius, Merz, Ruben, Feldmann, Anja, and Vazao, Teresa. Towards programmable enterprise wlans with odin. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, pp. 115–120.

[56] Su, M., Bergesio, L., Woesner, H., Rothe, T., Kpsel, A., Colle, D., Puype, B., Simeonidou, D., Nejabati, R., Channegowda, M., Kind, M., Dietz, T., Autenrieth, A., Kotronis, V., Salvadori, E., Salsano, S., Krner, M., and Sharma, S. Design and implementation of the {OFELIA} {FP7} facility: The european openflow testbed. *Computer Networks 61*, 0 (2014), 132 – 150. Special issue on Future Internet Testbeds Part I.

[57] Van Adrichem, Niels LM, Doerr, Christian, Kuipers, Fernando, et al. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE* (2014), IEEE, pp. 1–8.

[58] Wang, Guohui, Ng, TS, and Shaikh, Anees. Programming your network at runtime for big data applications. In *Proceedings of the first workshop on Hot topics in software defined networks* (2012), ACM, pp. 103–108.

[59] Yazici, Volkan, Sunay, M. Oguz, and Ercan, Ali O. Controlling a software-defined network via distributed controllers. *CoRR abs/1401.7651* (2014).

[60] Yonghong, Fu, Jun, Bi, Jianping, Wu, Ze, Chen, Ke, Wang, and Min, Luo. A dormant multi-controller model for software defined networking. *Communications, China 11*, 3 (March 2014), 45–55.

[61] Zink, Michael, Künzel, Oliver, Schmitt, Jens, and Steinmetz, Ralf. Subjective impression of variations in layer encoded videos. In *Quality of Service?IWQoS 2003*. Springer, 2003, pp. 137–154.

[62] Zinner, Thomas, Jarschel, Michael, Blenk, Andreas, Wamser, Florian, and Kellerer, Wolfgang. Dynamic application-aware resource management using software-defined networking: Implementation prospects and challenges. In *Network Operations and Management Symposium (NOMS), 2014 IEEE* (2014), IEEE, pp. 1–6.