Constraint-Aware Software-Defined Network for Routing Real-Time Multimedia

Oluwaseyi Oginni
Centre for Cloud Computing
Birmingham City University
Oginni.Oluwaseyi@bcu.ac.uk

Peter Bull
Centre for Cloud Computing
Birmingham City University
Peter.Bull@bcu.ac.uk

Yonghao Wang
Centre for Digital Media Technology
Birmingham City University
Yonghao.Wang@bcu.ac.uk

ABSTRACT

Traditional Ethernet-based IP networks do not have the capability to provide the Quality of Service (QoS) required for professional real-time multimedia applications. This is because they operate on a best-effort network service model that does not provide service guarantee. Network operators and service providers require a novel network architecture to efficiently handle the increasing demands of this changing network domain. Software-Defined Networking has emerged as an effective network architecture that decouples the control plane and data plane, which makes it capable of handling the dynamic nature of future network functions and intelligent applications while reducing cost through simplified hardware, software, and management.

This paper presents an SDN architecture for real-time low latency applications that offer adaptive path provisioning based on the calculated end-to-end delay, available bandwidth, and traditional shortest path first algorithm. The SDN architecture utilises the Ryu OpenFlow application programming interface (API) to perform real-time monitoring to collect network statistics and computes the appropriate paths by using this information. The experiment to ascertain the feasibility and evaluate the effectiveness of this approach is carried out in an emulated network environment using Mininet.

CCS CONCEPTS

• **Computer systems organization → Real-time system architecture**;

KEYWORDS

OpenFlow, Software Defined Networking, Controller, API, Real-time

ACM Reference format:
Oluwaseyi Oginni, Peter Bull, and Yonghao Wang . 2017. Constraint-Aware Software-Defined Network for Routing Real-Time Multimedia . In *Proceedings of The 15th International Workshop on Real-Time Networks , Dubrovnik, Croatia, June 2017 (RTN'2017),* 6 pages.

1 INTRODUCTION

Multimedia applications such as professional video and audio often require stringent Quality-of-Service (QoS). In order for a network to satisfy these requirements, it must possess the ability to reserve required resource and exert network control. The Internet Engineering Task Force (IETF) has developed several QoS architectures such as IntServ [28] and Diffserv [29], but these technologies have only had limited success because they rely on a distributed architecture where every network device is configured to run complex protocols [6]. These distributed network protocols such as Open Shortest Path First (OSPF) [14] and Enhanced Interior Gateway Routing Protocols (EIGRP) [25] do not readily support path provisioning based on network parameters such as end to end latency and available bandwidth. Although EIGRP was designed with this capability, in practice EIGRP only uses the lowest interface bandwidth along a path and cumulative interface delay for path selection by default [25].

The key element missing from existing QoS architectures is that they lack the ability to effectively support constraint-based routing, because most routing protocols are only able to support a single metric such as hop-count, or bandwidth to compute the shortest-path. In order for existing routing protocols to support a vast range of QoS requirements, they need to be re-developed with a more complex architecture, where the network is characterised with multiple metrics such as end-to-end delay, hop-count and available bandwidth. This is, however, not practical as traditional routing protocols are already reaching the limit of feasible complexity [27]. Although Multiprotocol Label Switching (MPLS) provides a partial solution through its ultra-fast switching and traffic engineering capability, it does not support multiple constraints and it is also very complex to deploy [23].

As a result of these problems, it has become necessary to have a QoS framework capable of providing the required QoS for multimedia data without further complicating network control protocols. Software-Defined Networking (SDN) is a new architectural framework for networking, developed to promote innovation and facilitate programmatic control of network devices by separating the control and data plane. The decoupling of the control plane and data plane enable easier deployment of new features and applications, easy network virtualisation and management, and centralisation of various middle-boxes into software control. Instead of implementing network policies and running complex protocols on each network device, the network is simplified into a forwarding hardware and the decision-making network controllers [11]. This emerging technology will reduce cost and promote flexibility in network operation while facilitating innovative network service delivery models. The deployment of middle-box functions as Virtual Network Functions allows network services (e.g., firewall and Network Addresses Translation (NAT)) to be provisioned as virtual devices and to be controlled as separate service components. An

RTN'2017, Dubrovnik, Croatia

SDN controller can also expose a Northbound Interface (NBI) that allows applications to directly coordinate service deployments with the establishment of data delivery routes while satisfying individual application service requirements [11], [4]. The primary benefit of this evolving trend is the opportunity of adopting the flexibility and interoperability obtained in cloud computing in the networking domain [4]. As a result, new architectural paradigm and techniques can be developed for the dynamic control and automation of both network and application services.

This paper presents a Constraint-Aware SDN architecture for multimedia applications that offers adaptive path provisioning based on end-to-end delay, available bandwidth and traditional shortest path first algorithm. The controller is designed to monitor network devices in real-time and select best path based on the application requirement of a particular flow. The controller consists of four main modules: Network monitoring, latency measurement, topology discovery and routing module.

The remainder of the paper is organised as follows: Section II provides a brief overview of SDN and OpenFlow. Section III provides a detailed explanation of the controller design and operation. Section IV explains the methodology, experimentation and analysis of the result. Finally section V concludes the paper and provide recommendations.

2 BACKGROUND

The advent of diverse types of real-time applications such as video streaming, online gaming, video conferencing etc. has greatly increased the complexity of network management systems. These applications impose different QoS requirements such as latency, delay and jitter. In addition, the continuous expansion of networks has further increased these network challenges. The existence of these diverse types of traffic on the network further imposes performance constraints on the underlying network infrastructure. As a result of these, network management systems are constantly challenged to satisfy these ever growing requirements while conforming to resource constraints [1]. There have been various research into dynamic QoS in IP networks [13] [17] [8] [15]. However, most of these approaches are either based on DiffServ [3] or RSVP [28], which inherits the problems of the underlying switching fabric [1].

SDN is a novel architectural framework that decouples the control and data planes, logically centralises the network intelligence and state, and abstracts the underlying network infrastructure from the applications. As a result, enterprises and carriers acquire remarkable programmability, automation, and network control, allowing them to construct extremely scalable, flexible networks that can quickly adjust to changing business needs [18].

Figure 1 shows a simplified representation of the SDN architecture, which consists of three layers. The infrastructure layer, also known as the data plane consists of the network elements. The data plane is responsible for forwarding data, as well as monitoring local information and collecting statistics [2].

The control layer, also known as the control plane, is responsible for programming and managing the forwarding plane. Therefore, it utilises the information supplied by the forwarding plane and specifies network operation and routing. It is made up of one or more controllers that communicate with the network elements through

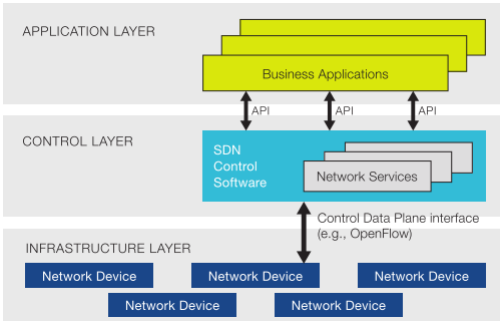

Figure 1: Software-Defined Network Architecture [18]

standardized interfaces, which are referred to as southbound interfaces [2].

The application plane consists of one or more applications, each of which has restricted control of a set of resources exposed by one or more SDN controllers [19]. The interface between the application layer and the control layer is known as the northbound interface [2]. SDN architecture provides a programmable access to the network elements through applications built on top of the control layer.

There have been previous attempts to provide an SDN framework that supports dynamic QoS for multimedia applications. OpenQoS [6] is an innovative controller design that facilitates QoS for multimedia delivery over OpenFlow networks. In order to support QoS, incoming traffic is grouped as data flows and multimedia flows, where the multimedia flows are dynamically allocated to QoS guaranteed routes and the data flows remain on their traditional shortest-path. OpenQoS provides an extension to the standard OpenFlow controller which enables it to support multimedia delivery with QoS. [5], also proposed the concept of a source-timed flow change, a technique for switching real-time packetised synchronous video streams using SDN and Openflow. In order to achieve this, a specific element of the packet header such as the UDP source port value is selected as the timing signal match field whose value is used to trigger a precise flow change by matching pre-configured SDN flow rules in the network. However, these approaches do not support path selection based on delay.

3 CONSTRAINT-AWARE CONTROLLER IMPLEMENTATION

OpenFlow provides the flexibility to associate a set of actions and rules to different types of flows. The Constraint-Aware controller is designed to route traffic flows based on application requirement, for instance, low latency flows will be routed based on delay while other flows may be routed based on available bandwidth or shortest path. The Constraint-Aware controller consists of four modules: Network monitoring, latency measurement, topology discovery and routing module. Each of these modules performs separate functions and work independently of each other: The monitoring module collects the necessary statistical information required to compute the available bandwidth of the links and paths in the network, latency measurement module computes the end-to-end delay of paths in the network, topology discovery module provides information on

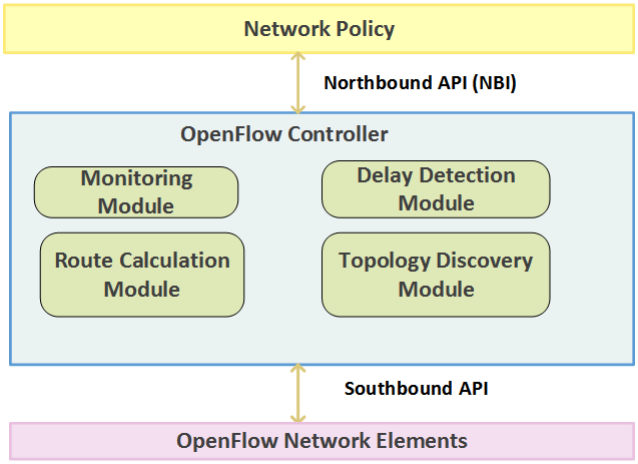

Figure 2: Controller Design

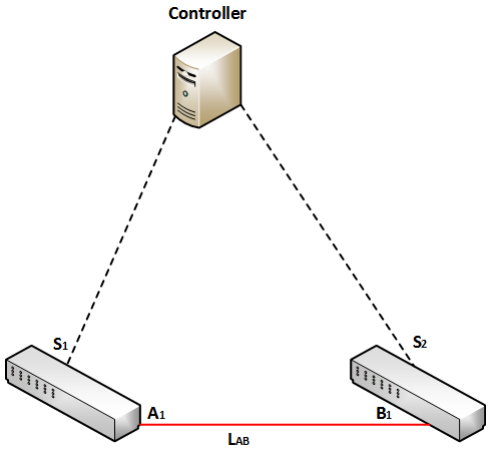

Figure 3: Bandwidth Calculation Scenario

devices and links in the network and the routing module computes the best path and installs the required flow rule on the devices along the path.

The Constraint-Aware controller proposed within this paper is implemented using the Ryu OpenFlow controller. There are other OpenFlow controllers such as Beacon, Maestro, OpenDaylight, Floodlight that can be used to implement the Constraint-Aware controller, Ryu was chosen because it provides software components with well-defined API that make it easy for network programmers to develop new network management and control applications. Ryu also supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. [24]. Figure 2 shows the schematic diagram of the proposed constraint-aware controller design.

3.1 Network Monitoring Module

Software-Defined Networks makes use of two monitoring techniques to provide a global view of the network: packet_in messages and per-port/per-flow counters. When there is no match for a packet in the flow table, the switch sends a packet-in message containing the packet header or part or the entire payload to the controller for processing. Therefore, in a typical setup, the controller receives a packet-in message at the start of every flow [26].

OpenFlow switches also maintain counters for each flow table, flow entry, port, queue etc. OpenFlow counters can be implemented in software and maintained by polling hardware counters with more limited ranges. Table 1 shows the per flow table, per flow entry and per port set of counters specified in the OpenFlow specification [22]. The network monitoring module makes use of the OpenFlow port statistic request and reply messages to collecting information from the switches. In other to send flow and port statistical request to the switches, Ryu makes use of OFPFlowStatsRequest and OFPPortStatsRequest messages to request for the corresponding statistical information from OpenFlow switches [24]. OpenFlow switches respond to this request by sending the corresponding reply messages. This message consists of a list of the statistical information requested. OFPPortStats response consists of statistical information such as port numbers, send and receive packet count, respectively, byte count, drop count, error count, frame error count, overrun count, CRC error count, duration(time port has been alive), and collision count [20]. This information is used in calculating the available bandwidth on all connected links in the network. In order to calculate the available bandwidth, the controller is programmed to sends OFPPortStatsRequest request at a specific interval to request port-related statistical information from switches. The value of the duration which is measured in seconds and nanoseconds beyond the duration in seconds [20], and transmitted bytes count (Tx) are saved at specific intervals. The controller calculates the available bandwidth of all links in the network through the process described below.

Given the network shown in Figure 3, available bandwidth on link L_{AB} can be calculated using the following parameters:

- Time taken to transmit = T
- Number of transmitted byte on Port $A_1 = Tx$
- Number of received byte on Port $B_1 = Rx$
- Total Port Bandwidth of $A_1 = A_{1total}$
- Total Port Bandwidth of $B_1 = B_{1total}$
- Used bandwidth on port $A_1(U_{A1}) = \frac{\Delta Tx}{T}$
- Used bandwidth on port $B_1(U_{B1}) = \frac{\Delta Rx}{T}$
- Available Bandwidth on port $A_1(BW_{A1}) = A_{1total} - U_{A1}$
- Available Bandwidth on port $B_1(BW_{B1}) = B_{1total} - U_{B1}$
- Available bandwidth on link $L_{AB} = Min(BW_{A1}, BW_{B1})$

Note that Tx should ideally be the same as Rx if the two switches are directly connected. In this application, the minimum between the two values is chosen for accuracy. The lowest link bandwidth across a path is chosen as the highest available bandwidth for that path.

3.2 Topology discovery

A reliable topology discovery mechanism is critical to the efficiency of SDN systems. It supplies necessary information needed by the controller to manage and provide services such as routing In the network [21]. OpenFlow switches do not have the functionality for

Counter	Bits	
Per Flow Table		
Reference Count (active entries)	32	*Required*
Packet Lookups	64	*Optional*
Packet Matches	64	*Optional*
Per Flow Entry		
Received Packets	64	*Optional*
Received Bytes	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Port		
Received Packets	64	*Required*
Transmitted Packets	64	*Required*
Received Bytes	64	*Optional*
Transmitted Bytes	64	*Optional*
Receive Drops	64	*Optional*
Transmit Drops	64	*Optional*
Receive Errors	64	*Optional*
Transmit Errors	64	*Optional*
Receive Frame Alignment Errors	64	*Optional*
Receive Overrun Errors	64	*Optional*
Receive CRC Errors	64	*Optional*
Collisions	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*

Table 1: List of Counters [20]

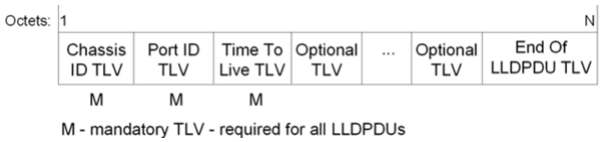

Figure 4: LLDP Frame Structure [10]

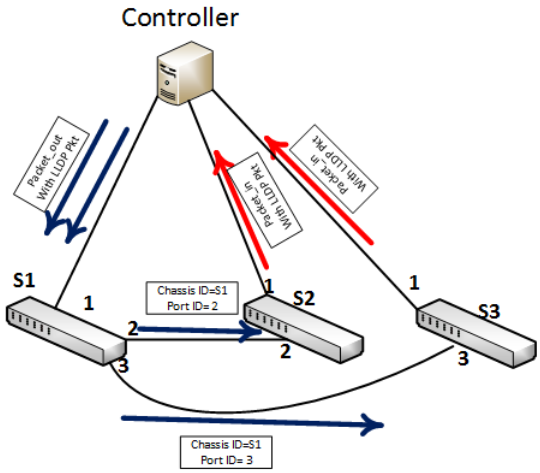

Figure 5: Topology Discovery Scenario

topology discovery, therefore it is the responsibility of the controller to provide this service. Furthermore, no standard has been defined yet for topology discovery in OpenFlow-based SDNs. However, topology discovery is implemented by individual controller platforms in a similar fashion to the implementation in NOX [17]. This mechanism, referred to as OpenFlow Discovery Protocol (OFDP) [7] has, therefore, become the de-facto standard for topology discovery in OpenFlow-based SDN [21]. OFDP leverages the Link Layer Discovery Protocol (LLDP) [7] . LLDP is a link layer protocol that allows an IEEE 802 Local Area Network (LAN) station to advertise its status and capabilities to other stations on the LAN. LLDP packets are transmitted to a link-local multicast MAC address (01:80:C2:00:00:0E), therefore only directly connected switches receive these advertisements. The information sent and received in each LLDP data unit (LLDPDU) is stored in one or more management information base (MIB) on the participating switches. The LLDP packet is encapsulated in an Ethernet frame with an EtherType value of 0x88cc. The frame consists of an LLDPDU, which contains an array of information elements, with each having a type, length and value (TLV) field. Each LLDPDU contains three required TLVs. The required TLVs are Chassis ID, Port ID and Time to live followed by other optional TLVs and an End of LLDPDU TLV. Figure 4 illustrates the frame structure

In OpenFlow-based SDN, the controller initiates the transmission of LLDP messages. The process is described in the scenario shown in Figure 5. OpenFlow switches have a pre-installed rule in the flow table with an instruction to forward any LLDP packet received from any port other than the controller port to the controller via a PACKET-IN message. The controller creates LLDP packet for every active port and instructs the switch to send it out these ports using a *PACKET_OUT* message. In this scenario, the controller creates LLDP packets for ports 2 and 3 on switch S1. The controller then sends these two packets to S1 and instructs S1 to send it out port 1 and 3 using separate *PACKET_OUT* messages. S2 and S3 then forward the received LLPD packet according to the pre-installed rule to the controller. The controller receives the packet_in message from the switches and process the information contained in the payload of the LLDP packet. From this information, the controller knows that a link exists between (S1, Port2) and (S2, Port2), and adds this information to the topology database. This process is repeated for all active OpenFlow switches connected to the controller.

The topology discovery module makes use of the Ryu rest topology API to get topology information. The two methods used in this module are the get_switch and get_link methods. These methods provide switch and link information respectively. This information is then computed and the graph of the entire topology is created using Networkx [16] python library.

3.3 Delay Detection Module

The network delay detection module utilises Ryu controller's LLDP API to obtain the timestamp of LLDP packets being transmitted. The process is described in the scenario shown in Figure 6. The controller sends an LLDP packet to S1 with a send timestamp. S1 sends the LLDP packet to S2 and S2 to the controller. The controller then calculates the total delay (Ttotal) from the controller back to the controller by subtracting the "send time" from the "receive time" of the LLDP packet. In order to calculate the delay between controller and switch, the controller sends an OFPEchoRequest

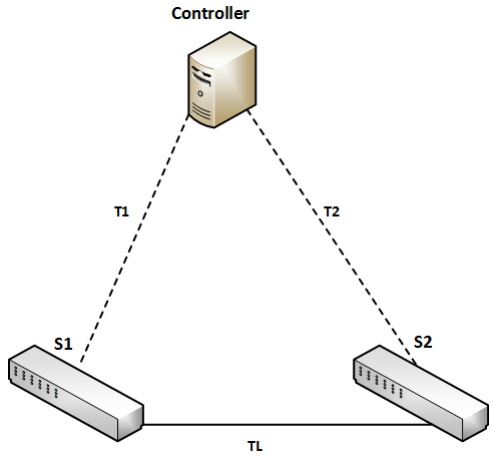

Figure 6: Delay Detection Process

with data being send time. The switch replies with OFPEchoReply message with the same data.

Therefore the delay between controller and switch is:
$\frac{echo\ receive\ time - echo\ send\ time}{2}$

With T_1 being delay between controller and S1, T_2 delay between controller and S2, and T_{total} total delay. The delay between S1 and S2 (T_L) can be expressed as follows:

$T_L = T_{total} - T_1 - T_2$.

The delay is computed periodically at a specified interval and saved in the topology graph. This information is then used to calculate the best path by delay.

3.4 Routing Module

The Routing module utilises the Networkx [16] Python package to compute shortest paths. NetworkX can be used for creating, analysing and manipulating networks and network algorithms. It also includes packages that provide data structures for interpreting different types of networks, such as simple graphs, directed graphs, and graphs with parallel edges and self-loops. In addition to these data structures, various graph algorithms are available for calculating network properties and structure measures: shortest paths, betweenness, centrality e.t.c [9]. When the controller receives a packet_in message from a switch in the network, the routing module processes the packet_in message and extracts the source and destination IP address. It then checks for the best path to the destination, if it already exists it uses the existing path. If it does not find a path it computes the path and saves it. The routing module then uses the topology information from the topology module to install the required flow rules on the switches along the path from source to destination.

4 TESTING AND RESULTS

The test network is created using Mininet [12] network emulator. The network consists of 4 OpenFlow switches, 6 hosts and a controller. In order to test the performance of the Constraint-aware controller, the test network is designed with 3 different paths, each

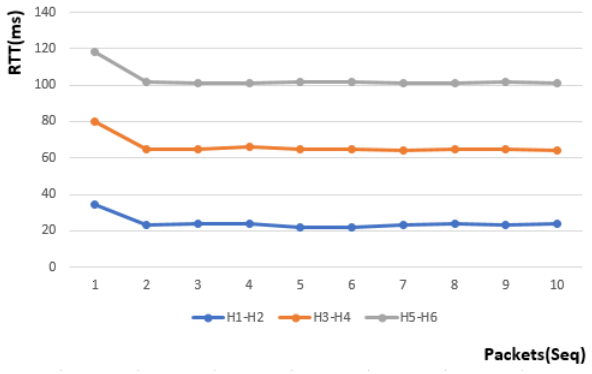

Figure 7: RTT results

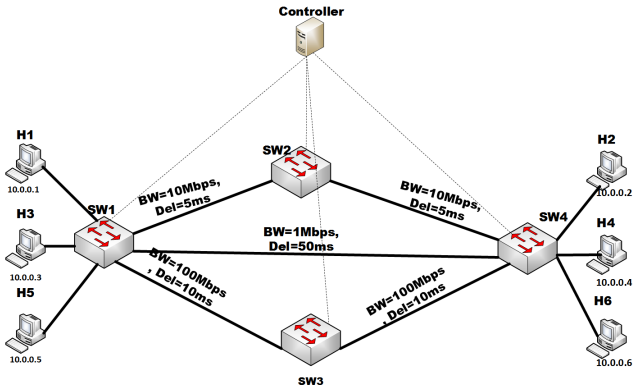

Figure 8: Test Topology

having different delay and bandwidth. It is important to note that Mininet does not have the ability to report the set bandwidth to the controller because the bandwidth is set in kernel space and Mininet is not aware of it. Therefore for this experiment, the available bandwidth is manually configured on the controller. Also in order for the delay module to accurately calculate the delay, all network devices and the controller's clock need to be synchronised. However, synchronisation is not required in this experiment because all devices on Mininet share the same clock with the host computer. The network is configured as shown in figure 8. The controller is designed with the capability to compute the best path using bandwidth, delay and hop count depending on the application requirement. For these experiments, the controller is configured to match flows based on their source IP address. The controller computes the part for H1 (10.0.0.1) using the path with the lowest delay, H3 (10.0.0.3) uses path with the highest available bandwidth and H5 (10.0.0.5) takes the path with the shortest path.

Figures 7 shows the round trip time (RTT) for the communication between different hosts. When H1 pings H2, the average RTT is 24.29ms, which indicates that the computed best path is the path with the lowest delay, that is, path [SW1-SW2-SW3]. When H3 pings H4, the average RTT is 66.36ms which shows that the best

path is the path with the highest bandwidth, that is, path [SW1-SW3-SW4]. Finally when H5 ping H6 the average RTT is 103.1ms which shows that the path is the path with the shortest hop count, that is, path [SW1-SW4]. Note that the RTT of the first packet is considerably higher due to the time taken for the controller to compute the best path and install corresponding flow on the switches. This mechanism is designed to work for all types of network topology. However, as the size of the network increases, the delay due to real-time monitoring and time taken to compute best path will increase.

5 CONCLUSION AND FUTURE WORK

An adaptive path provision mechanism can play an important role in IP networks considering the growth of multimedia applications, such as video-on-demand, video streaming, online gaming etc. The constraint-aware controller presented in this paper is a novel approach for ensuring efficient use of resources in a multi-path network, thereby improving the Quality of Experience for users. Constraint-aware routing schemes not only guarantee QoS requirements but can also increase overall network performance. This paper provides a framework to simplify the deployment of multimedia systems. With this approach, network engineers will no longer need to go through the arduous task of implementing complex QoS configurations on multiple devices. They only have to specify network policy and the controller implements this policy by sending instructions to the OpenFlow devices. For instance, a network engineer can simply specify the delay requirement for an application and the controller provisions the link with the specified requirements.

Future research will focus on the design and implementation of further components of the constraint-aware controller. This will include a QoS module that will provide admission control and traffic shaping functions. Experimental testing will also be expanded to include tests for other network parameters such as jitter, delay and bandwidth utilisation. These experiments will also be executed with hardware devices and a performance comparison will be carried out against other constraint-aware traffic engineering techniques, such as MPLS traffic engineering.

REFERENCES

[1] Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. 2013. PolicyCop: An autonomic QoS policy enforcement framework for software defined networks. In *2013 Workshop on Software Defined Networks for Future Networks and Services-SDN4FNS 2013*. DOI: https://doi.org/10.1109/SDN4FNS.2013.6702548

[2] Wolfgang Braun and Michael Menth. 2014. Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. *Future Internet* 6, 2 (2014), 302–336. DOI: https://doi.org/10.3390/fi6020302

[3] Elwyn Davies, Mark A. Carlson, Walter Weiss, David Black, Steven Blake, and Zheng Wang. 1998. An Architecture for Differentiated Services. (1998). https://tools.ietf.org/html/rfc2475

[4] Qiang Duan, Yuhong Yan, and Athanasios Vasilakos. 2012. Service-Oriented Network Virtualization for Convergence of Networking and Cloud Computing in Generation Networks. *IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT* 9, 4 (2012), 373–392.

[5] Thomas G. Edwards and Warren Belkin. 2014. Using SDN to facilitate precisely timed actions on real-time data streams. In *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*. 55–60. DOI: https://doi.org/10.1145/2620728.2620740

[6] H.E. Egilmez and S.T. Dane. 2012. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks. In *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC)*. 1–8. http://ieeexplore.ieee.org/xpls/abs

[7] GENI. 2011. OpenFlow Discovery Protocol. (2011). http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol

[8] S. Giordano, S. Salsano, S. Van den Berghe, G. Ventre, and D. Giannakopoulos. 2003. Advanced QoS provisioning in IP networks: the European premium IP projects. *IEEE Communications Magazine* 41, 1 (jan 2003), 30–36. DOI: https://doi.org/10.1109/MCOM.2003.1166651

[9] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy 2008)*. 11–15.

[10] Institute of Electrical and Electronics Engineers. 2009. *IEEE Standard for Local and Metropolitan Area Networks– Station and Media Access Control Connectivity Discovery*. Technical Report February. 1–204 pages. DOI: https://doi.org/10.1109/IEEESTD.2009.5251812

[11] Marc Mendon, Marc Mendon, Bruno Nunes Astuto, Xuan Nam Nguyen, Katia Obraczka, Thierry Turletti A, and Marc Mendonca. 2014. A Survey of Software-Defined Networking : Past , Present , and Future of Programmable Networks. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS* 16, 3 (2014), 1617–1634.

[12] Mininet. 2017. Mininet: An Instant Virtual Network on your Laptop (or other PC). (2017). http://mininet.org/

[13] M Mirhakkak, N Schult, and D Thomson. 2000. Dynamic Quality-of-Service for Mobile Ad Hoe Networks. In *1st ACM international symposium on Mobile (MobiHoc)*. 137–138.

[14] J. Moy. 1998. (1998).

[15] E. Mykoniati, C. Charalampous, P. Georgatsos, T. Damilatis, D. Goderis, P. Trimintzios, G. Pavlou, and D. Griffin. 2003. Admission control for providing QoS in DiffServ IP networks: the TEQUILA approach. *IEEE Communications Magazine* 41, 1 (jan 2003), 38–44. DOI: https://doi.org/10.1109/MCOM.2003.1166652

[16] NetworkX Developer Team. 2014. Networkx. (2014). http://networkx.github.io/documentation.html

[17] Thi Mai Trang Nguyen, Nadia Boukhatem, and Guy Pujolle. 2003. COPS-SLS usage for dynamic policy-based QoS management over heterogeneous IP networks. *IEEE Network* 17, 3 (may 2003), 44–50. DOI: https://doi.org/10.1109/MNET.2003.1201476

[18] Open Networking Foundation. 2012. *Software-Defined Networking: The New Norm for Networks*. Technical Report. 1–12 pages. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[19] Open Networking Foundation. 2014. *SDN Architecture*. Technical Report 1. 1–68 pages. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR

[20] Open Networking Foundation (ONF). 2014. OpenFlow Switch Specification. (2014). https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[21] Farzaneh Pakzad, Marius Portmann, Wee Lum Tan, and Jadwiga Indulska. 2014. Efficient topology discovery in software defined networks. In *2014, 8th International Conference on Signal Processing and Communication Systems, ICSPCS 2014 - Proceedings*. DOI: https://doi.org/10.1109/ICSPCS.2014.7021050

[22] B Pfaff, B Lantz, and B Heller. 2012. *OpenFlow switch specification, version 1.3. 4*. Technical Report. 1–171 pages.

[23] Eric Rosen and Yakov Rekhter. 2016. BGP/MPLS IP Virtual Private Networks (VPNs). (2016). https://tools.ietf.org/html/rfc4364

[24] Ryu Development Team. 2015. *ryu Documentation*. Technical Report. 322 pages.

[25] Donnie Savage, James Ng, Steven Moore, Cisco Systems, and Donald Slice. 2016. Cisco's Enhanced Interior Gateway Routing Protocol. (2016). https://tools.ietf.org/html/rfc7868

[26] Junho Suh, Ted Taekyoung Kwon, Colin Dixon, Wes Felter, and John Carter. 2014. OpenSample: A low-latency, sampling-based measurement platform for commodity SDN. In *Proceedings - International Conference on Distributed Computing Systems*. 228–237. DOI: https://doi.org/10.1109/ICDCS.2014.31

[27] Zheng Wang and Jon Crowcroft. 1996. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications* 14, 7 (1996), 1228–1234. DOI: https://doi.org/10.1109/49.536364

[28] John Wroclawski. 1997. The Use of RSVP with IETF Integrated Services. (1997). DOI: https://doi.org/10.1017/CBO9781107415324.004 arXiv:arXiv:1011.1669v3

[29] Lixia Zhang, Robert Braden, Raj Yavatkar, Michael Speer, Peter Ford, John Wroclawski, Bruce Davie, Eyal Felstaine, and Fred Baker. 2000. A Framework For Integrated Services Operation Over Diffserv Networks. (2000). https://tools.ietf.org/html/rfc2998