QoS-based Routing over Software Defined Networks

Andrew Kucminski, Ahmed Al-Jawad, Purav Shah, Ramona Trestian
School of Science and Technology,
Middlesex University
London, UK
{MK1629, AA3512}@live.mdx.ac.uk, {p.shah, r.trestian}@mdx.ac.uk

***Abstract*—Quality of Service (QoS) relies on the shaping of preferential delivery services for applications in favour of ensuring sufficient bandwidth, controlling latency and reducing packet loss. QoS can be achieved by prioritizing important broadband data traffic over the less important one. Thus, depending on the users' needs, video, voice or data traffic take different priority based on the prevalent importance within a particular context. This prioritization might require changes in the configuration of each network entity which can be difficult in traditional network architecture. To this extent, this paper investigates the use of a QoS-based routing scheme over a Software Defined Network (SDN). A real SDN test-bed is constructed using Raspberry Pi computers as virtual SDN switches managed by a centralized controller. It is shown that a QoS-based routing approach over SDN generates enormous control possibilities and enables automation.**

Keywords—Quality of Service, Software Defined Networks, Prioritized Routing, Networking.

I. INTRODUCTION

Nowadays, it is hard to imagine end-user devices without Internet connection. Similarly, all the big organizations have their own computer network connected to other organizations networks. As this network of networks is growing in recent years with a truly incredible speed, several trends are driving users, organizations or network providers to develop new network architectures. According to 0, these trends could be devised into three main categories: increasing demand, increasing supply and complex traffic patterns. Increasing demands refer to trends that increase load on enterprise networks as well as the Internet such as: Internet of Things (IoT), Big Data, cloud computing and mobile data traffic. Increasing supply is caused by rising demands which leads to capacity expansion of network technologies, such as 4G or 5G over Wi-Fi. If an organization requires specific network behavior, an application can be developed according to specific needs. These applications can be specific to common networking functions like traffic engineering and security, Quality of Service (QoS), routing, switching, monitoring, virtualization and load balancing. QoS requirements forced on the network are extended as a result of the multitude of applications, and then network traffic load must be orchestrated in an increasingly sophisticated and agile way.

In a traditional network, operators configure each node individually by using Command-line Interface, but this option can be limited to the functionality already installed. Large networks can contain many nodes to configure or reconfigure to implement new routing policies and any single network entity is not aware of the whole network topology. Moreover, the traditional networking industry has been dominated by vendors with their proprietary management and solutions that sometimes fail to satisfy their customers' needs. In this context, the main goal for Software Defined Networks (SDN) and the OpenFlow technology is to separate the hardware from the control software layer enabling the network operators to build cheaper and easier to manage networks by allowing the network to be open and programmable. This implies that network automation as data traffic can be manipulated, diverted and adjusted regardless of routing protocols.

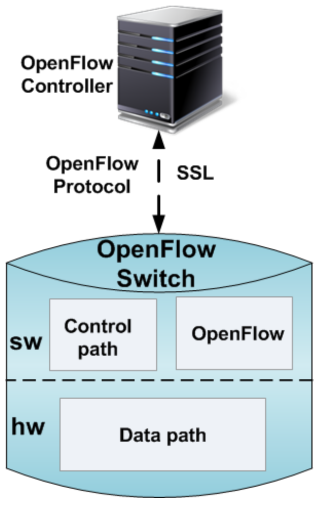


Fig. 1. OpenFlow Network

A traditional Ethernet switch consists of : data path – which represents the part dedicated to the hardware, responsible for packet forwarding; and the control path – which represents the part dedicated to the software, responsible for taking decisions, similar to an operating system.

In order to provide more control over the network, the OpenFlow enabled switch separates the control plane from the data plane. The control plane is moved outside the switch enabling remotely control of the data plane through a secure channel, as seen in Fig. 1. The control functions reside on the OpenFlow Controller, making them independent of the hardware they control. OpenFlow provides an abstraction of the data plane through the use of flow table that can be controlled over the secure channel by the OpenFlow Controller. By making use of OpenFlow controllers, the network administrators will be able to define flows and policies.

TABLE I. RELATED WORKS SUMMARY

Ref	Objective	Performance Metric	Evaluation Environment	Findings
[8]	Validate OpenFlow and test maximum throughput	Validate OpenFlow function. Maximum throughput with different segment sizes.	SDN testbed with Open vSwitch on Raspberry Pi	Similar performance to net-FGPA. OpenFlow functionality successfully operated.
[9]	Open vSwitch design and implementation.	High performance and optimization, flow caches	Hypervisor	Virtualization and flows controlling resulted with gradual optimatization for datacenters requirements workload.
[10]	Flow reconfiguration and efficient response to service demands through SDN.	Link up latency and link down latency.	SDN testbed called Pi Stack Switch	Network Administrator can recognize link state and topology changes in less than one second.
[11]	SDN management and configuration tasks across different network types.	Various types of policies such as: time, data usage, authentication status, and traffic flow.	Deployed in a campus network and a home network.	Procera is feasible for network policies, reduces the complexity of network management considerably for a range of network settings and various network policies.
[12]	Build and test OpenFlow based mirroring switch.	Throughput	Open vSwitch installed on laptop and on Raspberry Pi as mirroring switch.	OpenFlow-based laptop mirroring switches are more useful then Raspberry Pi for port mirroring.
[13]	Analytic Hierarchy Process (AHP) to select the best SDN controller.	Top five SDN controllers considering their current deployment and utilization.	Using Analytic Hierarchy Process (AHP) to select the best open source SDN controller.	Ryu is the best SDN controller taking into account the specific requirements.
[14]	SDN Controllers Performance Testing	Round Trip Time (RTT), TCP bandwidth	Mininet tree topology simulation	POX is not recommended for environments where performance is crucial.

In this context, the control of network traffic flows is moved from the infrastructure (switches and routers) to administrators.

Because evidently there are more and more users, devices and services on the network, imposing changes in the network infrastructure paves the way for new technologies such as SDN. In the industry, the benefits of SDN technologies can be seen in several networking sectors including: service providers, Enterprise Campus Infrastructure, Data Centre and Clouds and Wide Area Networks.

This paper investigates the use of QoS-based routing over SDN. A real experimental SDN test-bed using Raspberry Pi computers was built. The experiments demonstrate that QoS can be delivered not only by prioritizing some data traffic (e.g., multimedia streams) but also by redirecting particular traffic flows through different links aiming at optimal bandwidth utilization and fulfilling requirements based on valid policies.

II. RELATED WORKS

SDN has already produced remarkable interest from both, academia and industry. SDN with OpenFlow was first introduced by McKeown et al. in [2] as a promising way to enable innovation in production networks. Even though its first purpose was for researchers to run experimental protocols within their campus network, its advantages made it suitable for commercial networks, being adopted by major players in the market. For example, Google is using SDN with OpenFlow technology since 2010 in order to reduce the backbone network complexity and improve performance [3].

The SDN architecture can enable the dynamic QoS provisioning for various applications such as voice, video and even real-time communications. Its main advantage is that it simplifies monitoring and troubleshooting problems because it provides a high level of visibility of the service quality indicators, transmission of multimedia in real time and efficient and effective traffic management. Caba et al. [4] investigated QoS in the context of SDN showing a significant evaluation improvement and increase in traffic utilization in network throughput over bandwidth allocation and fairness among various traffic classes. The evaluation results also proved that actual QoS Config API is satisfying performance to enable dynamic configuration QoS on forwarding devices. SDN as relatively new technology is attractive to researches for tests under various environments. Araniti et al. [5] investigated the performance of SDN over wireless environments and concluded that the use of OpenFlow introduces benefits in terms of end-to-end delay, throughput, and jitter. Bayes' theorem and Bayesian network model were used in [6] to find the most feasible path that satisfies the QoS constraint. Whereas in [7], the authors propose a compression-based technique for SDN that aims at decreasing the link usage for QoS applications while increasing the network observability.

However, despite of testing SDN using virtual machines such as well know network emulator Mininet[1], test-beds that utilize low cost tiny computer machines with embedded open source Linux software can also be explored.

Kim et al. [8] tested the OpenFlow functionality over an experimental test-bed created using two Raspberry Pi with OpenVSwitch [9] and Floodlight[2] as the SDN controller. The authors highlighted the many benefits of the low complex experiment including effective performance comparable to

[1] Mininet - http://mininet.org/
[2] Floodlight - http://www.projectfloodlight.org/floodlight/

much more expensive devices, low cost and easy programmable. However, they assessed that Raspberry Pi with only one Ethernet interface is not sufficient to process multiple connection individually. Thus, Han et al. [10] developed another SDN experimental test-bed based on Raspberry Pi, created on Pi Stack Switch using the Network – Hypervisor called OpenVirteX. They addressed the issue of one Ethernet device by making use of USB to Ethernet adapters. This way the authors managed the network architecture with four Raspberry Pi computers, where one Raspberry Pi acted as a controller and three others as OpenFlow protocol supported switches. By use of OpenVirteX hypervisor with virtualization functionality they created Pi Stack Switch with 10 ports. The tests included topology changes latency, amount of time reaction due to a network failure and state changes of links.

As there is still a lot of on-going work within the OpenFlow technology, understanding its performance limitations is crucial before using it in large-scale deployments. A summary of related works is listed in Table I.

III. QoS-based Framework for SDN

A. System Architecture

The general QoS-based framework for SDN is illustrated in Fig. 2. The framework consists of a *SDN controller* integrating custom modules to manage routing, *SDN Switch* implementing the flow tables, and the end-host running the applications. The interaction between the SDN Controller and the SDN Switch is done using the OpenFlow Protocol.

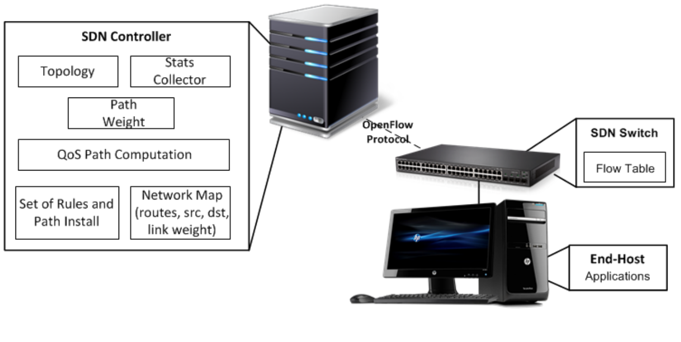


Fig. 2. System Architecture

The SDN controller consists of the following functional blocks:

- *Topology* block is used to store information about all devices, ports and the links currently up in the network. The topology discovery is done by generating link events using the Link Layer Discovery Protocol (LLDP) packets.
- *Stats Collector* block is used to keep track of the links load by periodically collecting load information from the switch ports counter.
- *Path Weight* block is used to compute a weight for each path in the topology. The weight is computed based on the available link capacity.
- *QoS Path Compute* block is used to compute the paths the flows could take to reach the destination and enable QoS provisioning.
- *Set of Rules and Path Install* block sends the computed paths and rules to the switch.
- *Network Map* block is used to store a map of the network.

B. QoS Path Computation

The QoS Routing is done based on a weighted routing algorithm as the one proposed in [15]. When the controller receives a packet of a new flow, it computes all the possible paths the flow might take to reach the destination. For each path, a weight is computed based on the current load. The path with the highest weight is selected as the target. The switch port counters are used to collect information about the load for each link, identify the imbalances in the traffic load and compute the paths weight.

Thus, assuming the network topology represented by a connected graph $G = (V, E)$, with the set of nodes V and the directed set of edges E. Given $F \subset V^2$ as the set of source-destination flows, a set $N_{(s,d)}$ of shortest paths between any source-destination pairs $(s, d) \epsilon F$ is computed. Furthermore, we assume that for any node $v \epsilon V$, any path $i \epsilon N_{(s,d)}$ has a number of M_i subpaths and each subpath $j \epsilon M_i$ has a number of S_{ij} segments. We denote with λ_{kji} and c_{kji} the traffic link load, which is taken from the SDN switch port counters by polling the switch port using standard OpenFlow mechanisms, and the link capacity on segment k, of subpath j, of path i, respectively. Thus, the *Link Utilisation Ratio (LUR)* is defined as in (1) [15].

$$LUR_{kji} = \frac{\lambda_{kji}}{c_{kji}} \quad (1)$$

Furthermore, at each node $v \epsilon V$, and for any path the flow could take to reach the destination, a weight is computed. Thus, for any path $i \epsilon N_{(s,d)}$ the weight w_i is computed using (2) [15] where $w_i \epsilon [0,1]$ and $\sum_i w_i = 1$. The highest the path weight, the less loaded the path is. The path is is selected as the target path and the set of rules and path install instructions are sent to the SDN switches on the path.

$$w_i = 1 - \frac{(\sum_j \sum_k LUR_{kji})/M_i}{\sum_i(\sum_j \sum_k LUR_{kji})/M_i} \quad (2)$$

IV. Experimental Setup

A. Experimental Test-Bed Setup

A real experimental test-bed was setup, as illustrated in Fig. 3, consisting of four Raspberry Pi running the Open vSwitch (OVS), two host PCs, a Server and the Open Network Operation System (ONOS)[3] Controller. Each Raspberry Pi is equipped with several USB to Ethernet adapters to enable multiple Ethernet ports on the OVS SDN switches and create a multi-path environment. The four OVS devices are managed by one centralized SDN controller ONOS installed on an Apple MacBook Pro machine running El Captian operating system. Table II presents a list with the hardware specification used for the experimental setup.

[3] ONOS - http://onosproject.org/

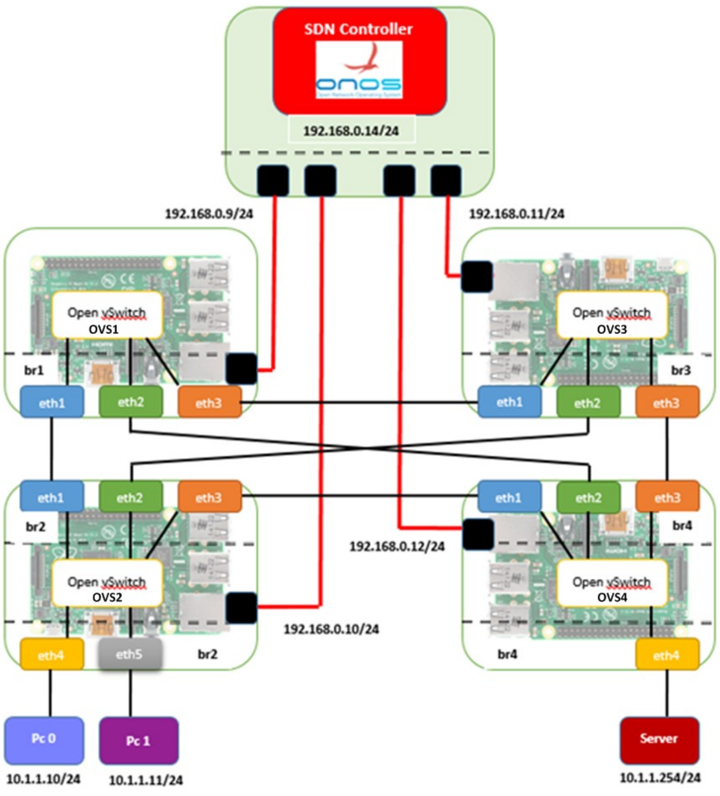


Fig. 3. Experimental Test-Bed

TABLE II. LIST OF HARDWARE COMPONENTS FOR EXPERIMENTAL TEST-BED

Hardware	Version	Specification
Raspberry Pi 2	Model B	Quad Core CPU 900 MHz, 1GB RAM, 16GB ROM
USB 2 Fast Ethernet Adapter	VK-QF970	RJ45 10
USB Fast Ethernet Adapter	D Link	RJ45 10/100
Network Cables	Cat 5e	Up to 100Mbps
Sony VAIO workstation	VPCJ 12 LOE	
HP Laptop	ProLiant Gen8 G1610T	
Apple Laptop	MacBook Pro 13	
Switch	NETGEAR GS305-100UKS	5-Port Gigabit

B. *Experimental Scenarios*

Several experimental scenarios are considered to test the performance of QoS-based routing over the real experimental test-bed, such as: continuous UDP traffic is generated between the PC0 host and the Server considering four data rates: 0.5 Mbps, 1Mbps, 2Mbps and 3Mbps. Whereas, TCP traffic is generated between the PC1 host and the Server. iPerf version 3 was used for generating traffic. In this case, we investigate the use of QoS-based routing and shortest path routing. Additionally, a link failover scenario is considered where the shortest path link is disturbed while PC1 host is transmitting data to the Server.

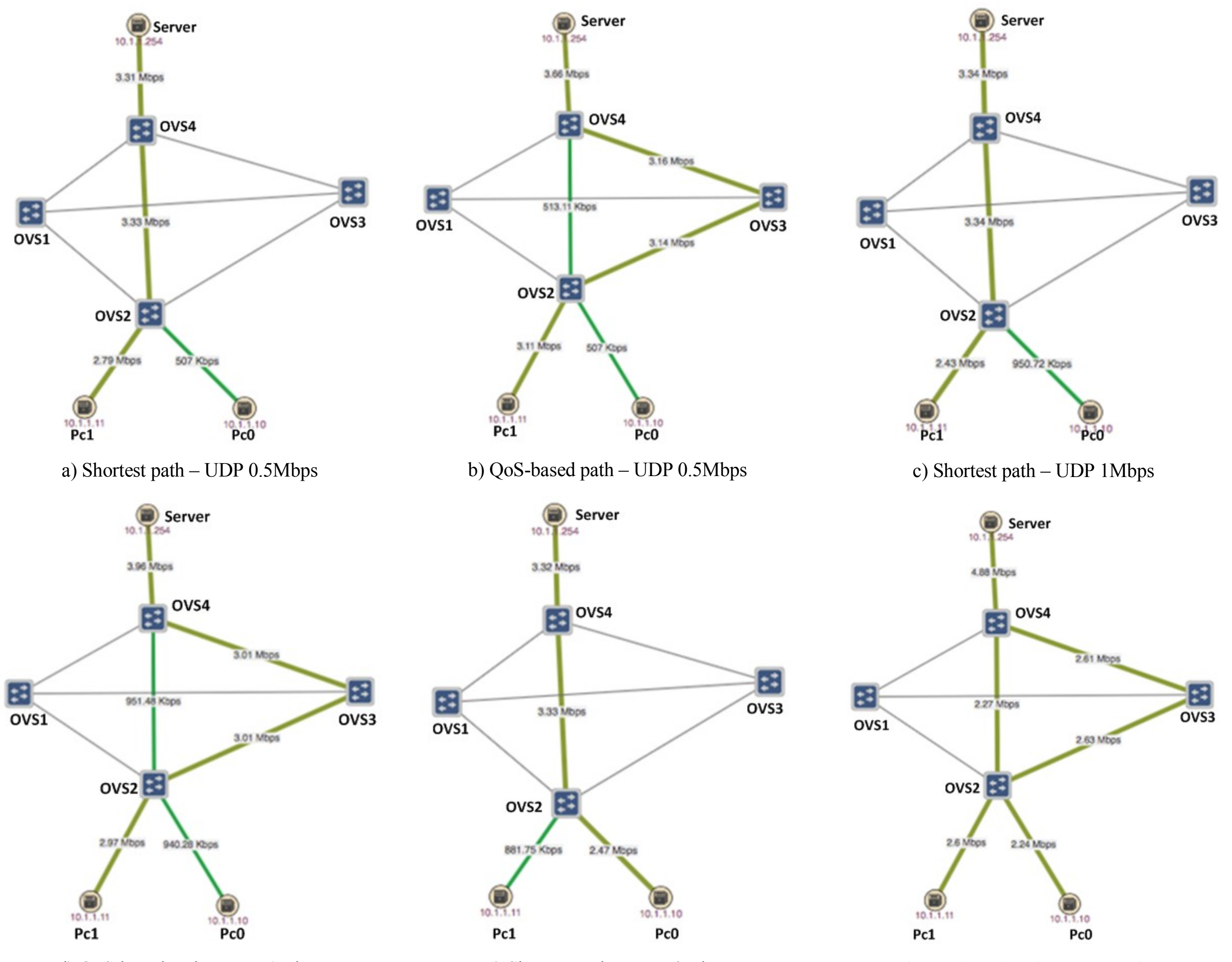
V. RESULTS AND DISCUSSIONS

A. *Test-Bed Limitations*

The overall test-bed setup was tested in order to determine the capacity and the maximum available resources. This was done using iPerf. The Raspberry Pi 2 model B is equipped with one RJ45 10/100 network socket and four USB ports. Thus, to enable us to create a mesh topology network four USB to Ethernet adapters were used for each Raspberry Pi. Although, Raspberry Pi has sufficient processing power to handle the connected USB to Ethernet adapters, the tested speed of the adapters turned out to achieve 4Mbps only, which limited the overall test-bed capacity. Thus, the maximum generated traffic in case of UDP was set at 3 Mbps.

B. *System Throughput*

The overall test-bed is monitored and managed by the ONOS controller which also enables us to view the network topology with all the devices and the active flows through the ONOS GUI. We investigated the maximum throughput that can be achieved by the system when using the shortest path and the QoS-based path. In the first case, all the traffic is routed over the shortest path following the OVS4-OVS2 link (see Fig. 3). In the case of QoS-based routing, as UDP starts first, it will occupy the shortest path. When the TCP traffic starts, the SDN controller will route it through the next less congested path, using the OVS4-OVS3-OVS2 links. In this way, the imbalance in the traffic load over the experimental test-bed is avoided. Fig. 4 illustrates the achieved throughput from the ONOS GUI for various data rates of the UDP traffic (i.e., 0.5Mbps, 1Mbps, and 3Mbps) and the TCP traffic between PC1-Server in case of



a) Shortest path – UDP 0.5Mbps b) QoS-based path – UDP 0.5Mbps c) Shortest path – UDP 1Mbps

d) QoS-based path – UDP 1Mbps e) Shortest path – UDP 3Mbps f) QoS-based path – UDP 3Mbps

Fig.4. Throughput for Shortest path and QoS-based path for UDP 0.5Mbps, 1Mbps, and 3Mbp

.

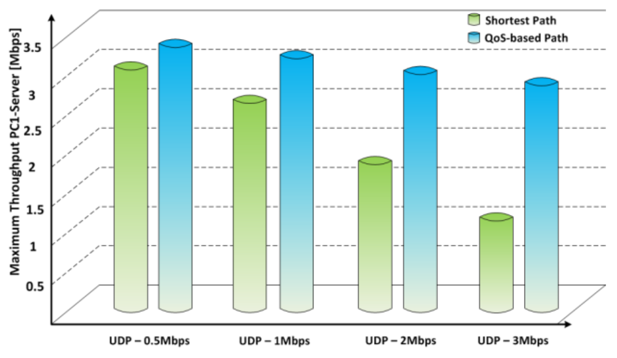


Fig. 5. Achieved Throughput PC1-Server under the four UDP data rates

shortest path and QoS-based path. The achieved throughput for all the UDP data rates is illustrated in Fig. 5. It can be noticed that in case of the shortest path, where the traffic flows are sharing common links, for the first scenario with UDP data rate of 0.5Mbps, the TCP traffic reaches as high as 2.79Mbps. However, as the UDP data rate increases the throughput for the TCP traffic decreases significantly, reaching as low as 0.88Mbps when the UDP data rate of 3Mbps is streamed. In the case of QoS-based path, as the UDP traffic increases it has a negligible impact on the TCP throughput. For example, when the UDP stream data rate is 0.5Mbps the TCP throughput goes as up as 3.11Mbps, whereas when the UDP stream data rate is 3Mbps the TCP traffic throughput goes as low as 2.6Mbps.

C. System Packet Loss

In order to investigate the overall system performance in terms of packet loss in case of shortest path and QoS-based path, UDP traffic was generated between PC1-Server at the maximum link capacity. The UDP traffic between PC0-Server was set at 1Mbps, 2Mbps and 3Mbps. In this way, the two paths containing the OVS4-OVS2 link and the OVS4-OVS3-OVS2 links are over utilized. Thus, in the case of shortest path all the traffic is going through the OVS4-OVS2 link, whereas in the case of QoS-based path, the PC0-Server UDP traffic is passing through the OVS4-OVS2 link and the PC1-Server UDP traffic is going through the OVS4-OVS3-OVS2 links. The results are illustrated in Fig. 6. As the test-bed is also limited by the capacity and processing of the USB to Ethernet adapters, this causes increased packet loss especially when the setup is used at its maximum capacity. Additionally, the packet loss when the shortest path is used is much higher than when the QoS-based path is used.

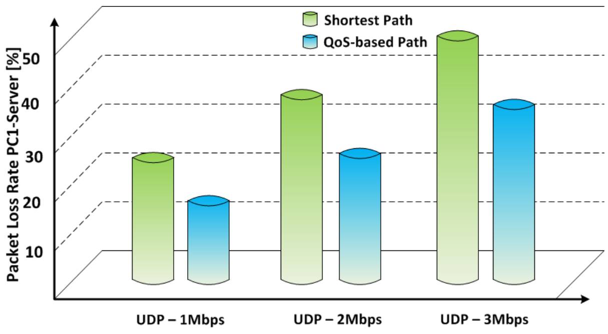
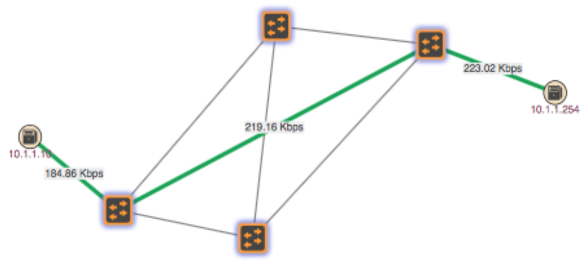


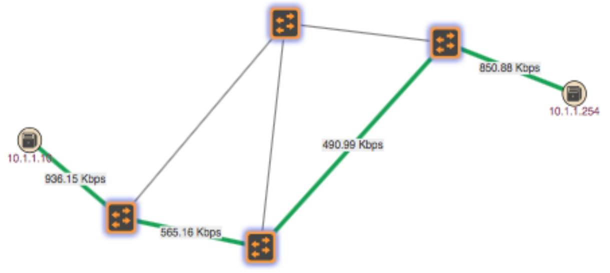
Fig. 6. Packet Loss Rate PC1-Server under three UDP data rates

D. Link Failover

In this scenario we test the reaction of the system to the link failover. To emulate the link failure, the cable connecting two switches that connect the hosts was removed, during data transmission as illustrated in Fig. 7. Initially the traffic is following the shortest path (see Fig. 7a) when the link drops. The ONOS controller detects the link failure and reroutes the traffic over a different path (see Fig. 7b). The latency introduced by the controller for link down is 3ms and for link up is 8ms. This latency is due to the time it takes for the switch to respond to OpenFlow *request* and *replay* messages.



a) Shortest Path Routing



b) Re-routing in case of Link Failover

Fig. 7. Link Failover Example

VI. CONCLUSIONS

This paper investigates the use of QoS-based routing over a controlled SDN environment. A real experimental test-bed was setup using Raspberry Pi computers running virtual SDN switches and enabling a multi-path SDN environment managed by the ONOS controller. The results show that compared to the shortest path routing, a QoS-based routing approach enables optimal bandwidth utilization by redirecting some data traffic through less congested links and avoiding the network traffic imbalances. However, the small scale test-bed is limited in performance by the USB to Ethernet adapters used which reduces in turn the overall system capacity.

Future works will consider creating a more stable test-bed setup and faster USB to Ethernet adapters will be used to improve the overall performance of the test-bed. A scalability scenario will also be considered.

REFERENCES

[1] W. Stallings, Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud. Addison-Wesley Professional, 2015.

[2] N. McKeown *et al.*, ‘OpenFlow: enabling innovation in campus networks’, *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, Mar. 2008.

[3] L. Faughnan, ‘Software Defined Networking’, *TechCentral.ie*, 01-May-2013. [Online]. Available: http://www.techcentral.ie/pro/22. [Accessed: 07-Jan-2017].

[4] CM. Caba, J. Soler, ‘SDN-based QoS Aware Network Service Provisioning’ in S Boumerdassi, S Bouzefrane & É Renault (eds), *Mobile, Secure, and Programmable Networking.* Springer, pp. 119-133. Lecture Notes in Computer Science, vol. 9395, DOI: 10.1007/978-3-319-25744-0_11

[5] G. Araniti, J. Cosmas, A. Iera, A. Molinaro, R. Morabito, and A. Orsino, ‘OpenFlow over wireless networks: Performance analysis’, in *2014 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, 2014, pp. 1–5.

[6] A. Al-Jawad, R. Trestian, P. Shah, and O. Gemikonakli, ‘BaProbSDN: A probabilistic-based QoS routing mechanism for Software Defined Networks’, in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–5.

[7] A. Al-Jawad, P. Shah, O. Gemikonakli, and R. Trestian, ‘Compression-based technique for SDN using sparse-representation dictionary’, in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 754–758.

[8] H. Kim, J. Kim, and Y.-B. Ko, ‘Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking’, Advanced Communication Technology (ICACT), 2014 16th International Conference on, 2014, pp. 758–761.

[9] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, “The Design and Implementation of Open vSwitch.” USENIX NSDI 2015.

[10] S. Han and S. Lee, ‘Implementing SDN and network-hypervisor based programmable network using Pi stack switch’, 2015, pp. 579–581.

[11] H. Kim and N. Feamster, ‘Improving network management with software defined networking’, *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[12] K. Ohira, ‘Performance Evaluation of an OpenFlow-based Mirroring Switch on a Laptop/Raspberry Pi’, in *Proceedings of The Ninth International Conference on Future Internet Technologies*, New York, NY, USA, 2014, p. 20:1–20:2.

[13] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, ‘Feature-based comparison and selection of Software Defined Networking (SDN) controllers’, in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 2014, pp. 1–7.

[14] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu, and E. C. Popovici, ‘A comparison between several Software Defined Networking controllers’, in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (SIKS)*, 2015, pp. 223–226.

[15] R. Trestian, G.-M. Muntean, and K. Katrinis, ‘MiceTrap: Scalable traffic engineering of datacenter mice flows using OpenFlow’, in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, 2013, pp. 904–907.