Providing of QoS-Enabled Flows in SDN *Exemplified by VoIP Traffic*

Jannis Ohms[1], Olaf Gebauer[1], Nadiia Kotelnikova[1,2], Diederich Wermser[1] and Eduard Siemens[2]
[1]*Research Group IP-Based Communication Systems, Ostfalia University of Applied Sciences, Salzdahlumer Str. 46/48, D-38302, Wolfenbüttel, Germany*
[2]*Future Internet Lab Anhalt, Anhalt University of Applied Sciences, Bernburger Str. 55, D-06366 Köthen, Germany*
{jannis.ohms, ola.gebauer, n.kotelnikova, d.wermser}@ostfalia.de, eduard.siemens@hs-anhalt.de

Keywords: Software-Defined Networking, SIP, QoS, OpenFlow.

Abstract: This paper provides a proof of concept of an SDN Application to provide QoS for real-time services on SDN networks. Common real-time services are for example VoIP or M2M protocols like OPC UA and MQTT. We provide a proof of concept in the form of a specialized application for SIP traffic. The application lowers the latency and call-setup-time for VoIP calls. The application uses the metering and queueing features of OpenFlow 1.3 to assure high quality of service. The evaluation and optimization of the application is still in progress.

1 INTRODUCTION

Software-Defined Networking or (SDN) for short is a new approach for the implementation of computer networks. SDN will be an integral part of the new 5G mobile network (Tsagkrais, 2015). SDN Networks consist out of multiple SDN switches and at least one SDN controller. The controller has an overview of the whole topology and creates flow rules for the switches. The controlled switches forward packets according to the flow rules they received. The primary used protocol for SDN is OpenFlow (Open Flow Network Foundation, 2012) which is specified by the Open Networking Foundation (ONF). Reactive forwarding increses the overall latency (Keqiang, et al., 2015). This could be curcumvented through the use of proactive forwarding which creates the requierd flows in advance. Low latency is a requirement of real-time applications like M2M communication or VoIP. Through the use of specially designed SDN applications, it could be possible to reduce the latency even further if the required flows would be provided proactively before the traffic needs to be forwarded. The required SDN application is related to the required protocol. In this paper, we are using the SIP (Session Initiation Protocol) (Rosenberg, et al., 2002) protocol as an example.

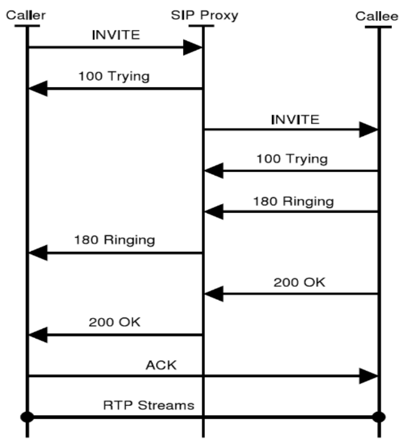

Figure 1: Sequence diagram of a SIP call setup with SIP proxy.

2 THE SIP PROTOCOL

SIP is a protocol used to implement the signaling in VoIP systems. The actual media data is transported by the RTP (Real Time Protocol) protocol (Schulzrinne, et al., 2003).

Figure 1 shows the call setup when a SIP proxy is used. The caller sends an invite message that is acknowledged with a trying response by the proxy. The proxy forwards the invite to the callee, which replies with a trying response. After that the call sends a RINGING message to the proxy to indicate that the called phone is ringing. The proxy forwards the RINGING message to the caller. The callee sends an OK message which contains the used codecs and media endpoints of the callee's telephone besides other information. The caller receives the OK message of the proxy and acknowledges it with an ACK message. After the ack message is received, the RTP stream for the media data is set up. To terminate a call, one party can send a BYE message, which is passed through the proxy and afterwards is acknowledged by an ok message.

3 BASIC IDEA OF THE SDN APPLICATION

The proposed application receives a copy of any SIP traffic on the network. This opens the opportunity to proactively push flows to establish a path for the RTP stream. The application can use the media description of the callee's OK message to make a bandwidth estimation and reservation to ensure a feasible quality of service. This happens proactively before the switches forward the actual RTP streams.

4 RELATED WORK

Adami et al. (Adami, et al., 2015) propose a special load-aware routing application to guarantee QoS for VoIP calls in Software-Defined Networks. The application measures the link utilization and chooses a path with low utilization for VoIP data. In contrary to that Egilmez et al. (Egilmez, et al., 2013) developed a special SDN controller which monitors the states of the different links in the topology to detect congested links. Walner et al. (Wallner & Cannistra, 2013) propose the use of OpenFlow queues in combination with ToS (Types of Service) Header fields. The disadvantage of the ToS field is that it has to be set at the client devices. Our application will provide QoS without the manipulation of the clients. Jeong et al. (Jeong, et al., 2012) developed a new NOS (network operating system) which uses network slicing and virtualization to enable QoS. Our approach combines active traffic analysis with OpenFlow queues. Our approach does not need to monitor the network state since the bandwidth needed is reserved by the queues. There are also approaches from within the field of industrial automation to use SDN for industrial automation. Herlich et al. (Herlich, 2016) use an open real-time Ethernet standard on regular SDN switches. Through the use of SDN, they achieve a more robust and flexible topology. SDN enabled them to dynamically change routes in the case of errors like link failures or broken switches. They achieved low latency and the required QoS through the use of a master node which enforced a strictly organized media access. The use of master nodes is fairly common in industrial Ethernet standards (Dürkop, et al., 2015). This forms a contrast to our approach since we would not rely on a central master or specific Ethernet based protocol.

5 DEVELOPMENT AND TESTING ENVIRONMENT

The testing and development environment for the application as seen in Figure 2 consists of the following:

- Two Snom VoIP phones representing user equipment
- One System running an instance of the Camalio SIP Proxy
- One System running an Instance of the Floodlight SDN controller
- Three Edgecore AS4610-30T bare metal switches running PicOS as operating system.

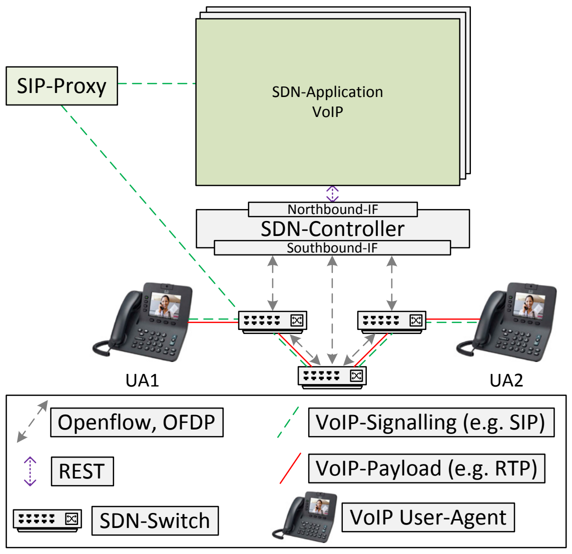

Figure 2: Testing a development environment for the VoIP-application.

6 LOGICAL ARCHITECTURE OF THE APPLICATION

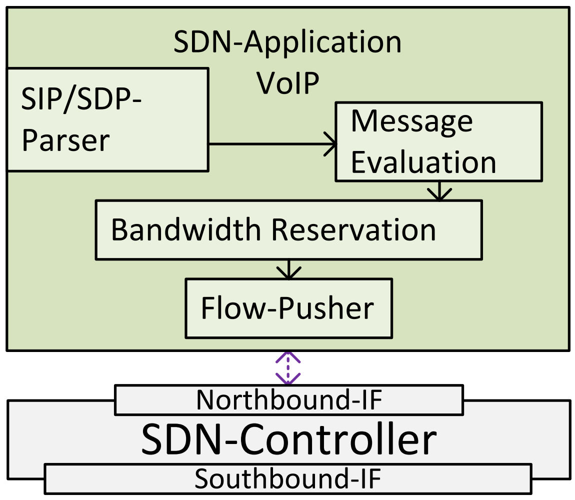

Figure 3: The internal components of the application on an abstract level.

The application consists of four components. The SIP/SDP (Handley, et al., 2006) parser is used to convert the received SIP packets into objects and to remove malformed packets. The Message Evaluation extracts the needed information for the bandwidth reservation and the routing. The information extracted from the Message Evaluation gets passed on to the bandwidth reservation which then estimates the bandwidth and assigns an OpenFlow queue for the call. The Flow-Pusher takes the callers and callees IP and calculates a path on the present topology. After the path is calculated all required flows get pushed. The Flow-Pusher also pushes a flow to every new switch which copies any SIP traffic and sends it to the controller.

This architecture provides the following main benefit: A clear separation between traffic processing analysis and path selection, which makes the components highly reusable and adaptable. This makes it possible to adapt the application for other protocols by providing a new analysis module and making minor changes like adjusting port numbers for the traffic processing. This architecture is visualized in Figure 3.

7 DEVELOPMENT OF THE SDN APPLICATION

Figure 4 shows the internal architecture of the Floodlight SDN controller (Project Floodlight, 2017). The controller consists of multiple independent and logical modules to provide a Java API as a northbound interface for user defined applications to use. Furthermore, the controller provides a REST and a Java API (Thomas, 2000) for external applications. The Java API provides huge performance benefits since the module can use all advantages of the controller like the module loading system and thread pools for concurrent execution. The proposed application will process any incoming SIP packets, so the usage of the Java API is advantageous for the performance. Floodlight provides a well gathered documentation and an easy understandable architecture which makes Floodlight a good project for prototyping and easy development. Our SDN-application consists of three modules. The first module is the SIPFlowPusher which receives incoming SIP traffic and sets up newly added switches to forward incoming SIP traffic to the controller. The second module is the SIPMsgAnalyzer which takes the SIP traffic from the SIPFlowPusher to extract the information. This is necessary to setup the required RTP streams. The RTPFlowPusher takes the extracted information to set up a path through the network and to choose the appropriate queue for a good quality of service.

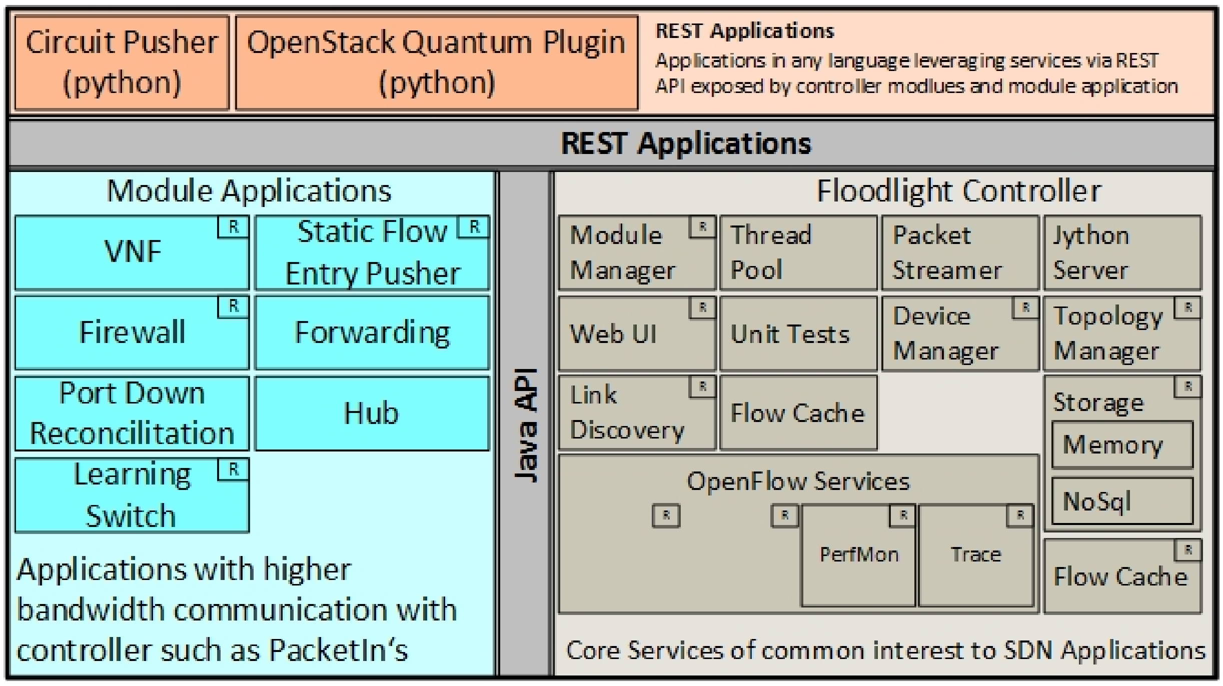

Figure 4: Architecture of the Floodlight SDN-controller (mod (Project Floodlight, 2017)).

The application uses the following services and interfaces of the controller

- **IFloodlightModule:** This interfaces is implemented by any of the three modules to indicate their status as a module to the rest of the system.
- **IOFMessageListener:** This interface enables the SIPFlowPusher module to receive incoming SIP traffic for further processing.
- **IOFSwitchListener:** This interface notifies the SIPFlowPusher when a new switch is connected to the network.
- **OFSwitchService:** This service enables a module to send OpenFlow messages to the switch to manipulate the content of the switches flow table.
- **RoutingService:** The RoutingService is used by the SIPMsgAnalyzer to determine a route for the RTP streams.
- **DeviceService:** The DeviceService is used to determine the physical port and MAC addresses of connected terminal devices.

The application also uses a third party library which is not part of the controller. The jain-sip library (O´Doherty & Ranganathan, 2017) provides a SIP parser for the application. The use of this library reduces the implementation time for the SIPMsgAnalyzer drastically since we don't have to write our own parser.

8 MEASUREMENTS

We performed the following measurements to examine if a bandwidth reservation can be provided by OpenFlow queues. Our measurement setup is shown in Figure 5. We connected two load generators to the switch; we also use a receiver to measure the amount of data received. The load generator LG1 generated one high priority stream of 700 Mbit/s. LG2 generated a low priority stream of 1 Gbit/s. The high priority stream has a duration of 10 seconds; the low priority stream has a length of 30 seconds. The high priority stream started roughly 10 seconds after the low priority stream. We compared the throughput of the different streams using different transport protocols (TCP and UDP) with different OpenFlow queue configurations. Figure 6 shows the use of TCP without queues. The bandwidth of the outgoing interface is shared equally between the two streams which implies the usage of a round robin algorithm. Figure 7 shows the use of TCP with queues in place. The queue is configured to reserve 700Mbit/s for the high priority stream. The graph indicates that the reserved bandwidth is provided. The Figure 8 and Figure 9 show the same results regarding UDP. The bandwidth could not be provided. One explanation for this could be the absence of flow control for UDP, this could create an overflow in the switches packet buffer which results in a drop of incoming packets. To further analyze the results regarding UDP we repeated our experiment with an overestimated queue size. We reserved 700 Mbit/s for a 100 Mbit/s stream. The results are shown in Figure 10. The results seem to imply that the size of the queue needs to be overestimated in order to work for UDP. This could also be an indicator that the throughput of the load generator and the throughput of the queue are measured on different OSI Layers.

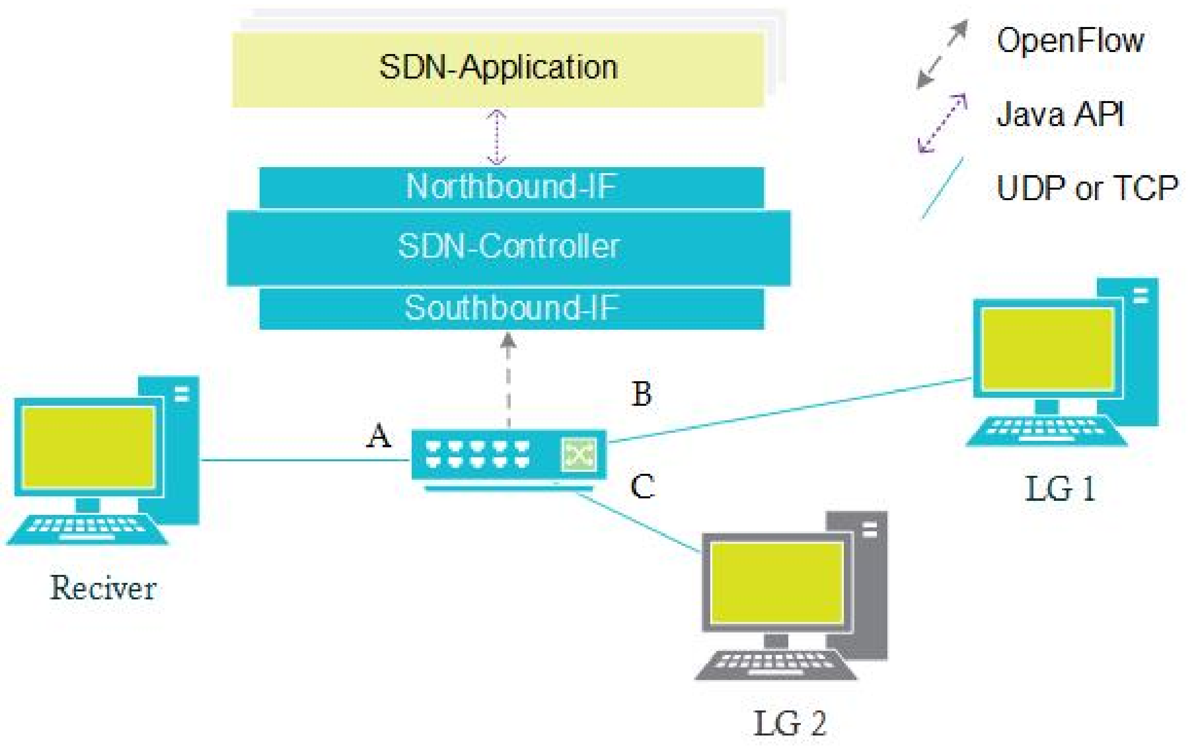

Figure 5: Measurement setup.

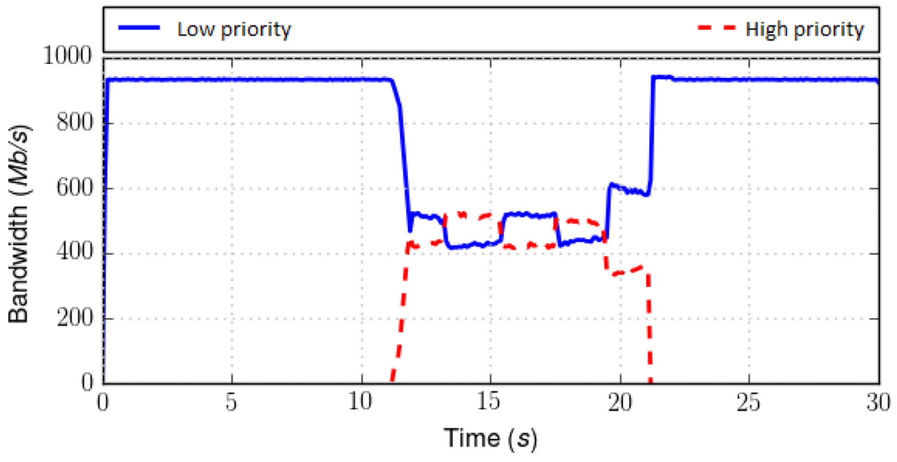

Figure 6: No bandwidth reservation, TCP.

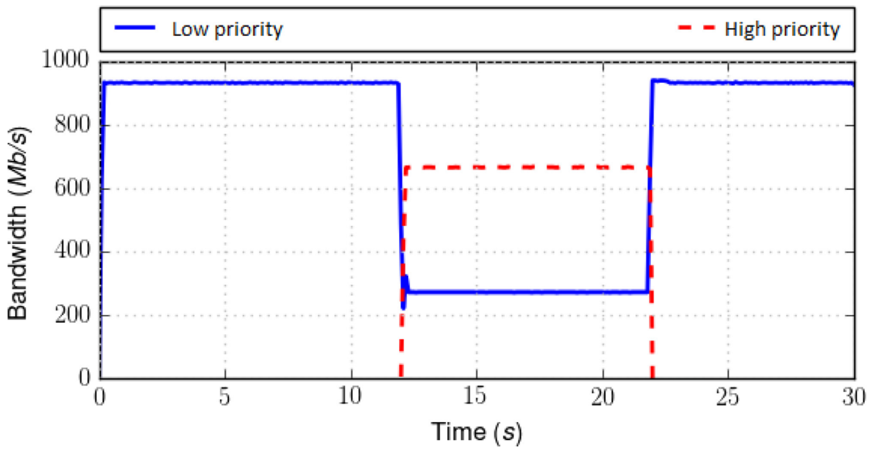

Figure 7: 700 Mbit/s bandwidth reservation, TCP.

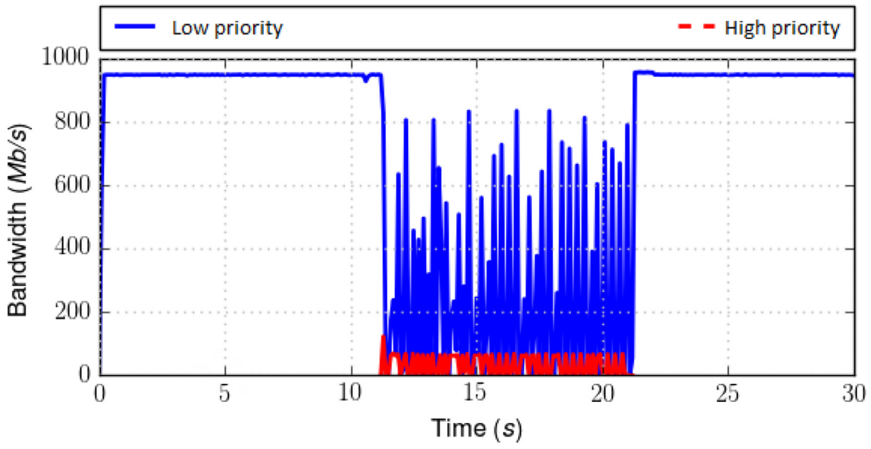

Figure 8: No bandwidth reservation, UDP.

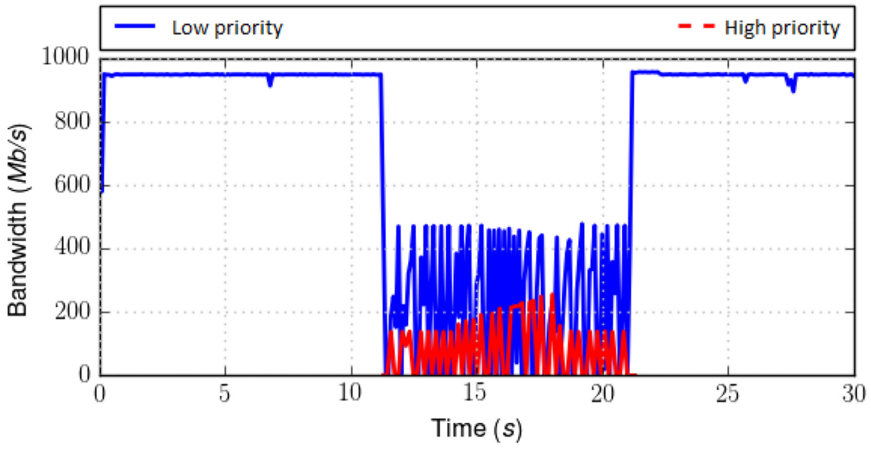

Figure 9: 700 Mbit/s bandwidth reservation, UDP.

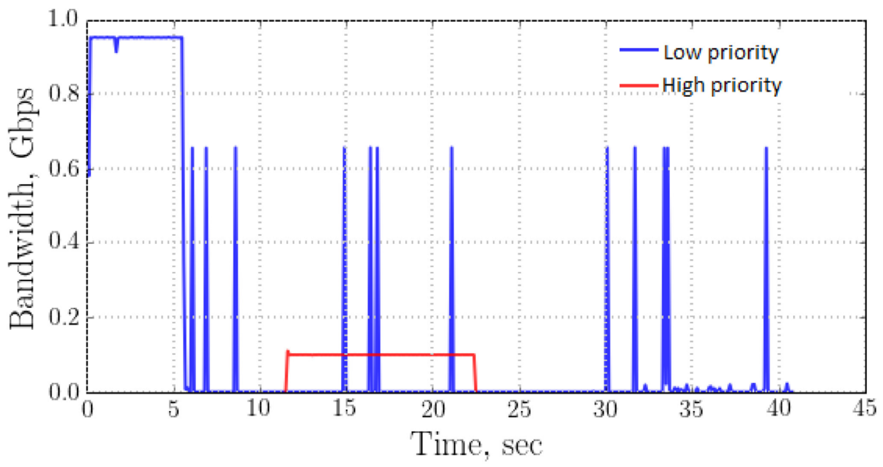

Figure 10: 100 Mbit/s high priority stream with a 700 Mbit/s bandwidth reservation, UDP.

9 CONCLUSIONS

The results show that a bandwidth for TCP streams can be guaranteed with OpenFlow queues. The results for UDP show that further research in the area of queue planning and capacity estimation needs to be done. Also, additional aspects such as jitter and delay of the streams need to be analysed.

ACKNOWLEDGEMENTS

The research presented in this paper is partly funded by the BMWi (Bundesministerium für Wirtschaft und Energie) within the ZIM-Programm (Zentrales Innovationsprogramm Mittelstand). This work is part of the INAASCA project (Integrated Network as a Service Solution as Part of Cloud IT Application Portfolio) (Gebauer, et al., 2016). Additionally, this work is funded by Volkswagen Foundation for trilateral partnership between scholars and scientists from Ukraine, Russia and Germany within the project CloudBDT: Algorithms and Methods for Big Data Transport in Cloud Environments.

REFERENCES

Adami, D., 2015. *Towards an SDN Network Control Application for Differentiated Traffic Routing - IEEE 978-1-4673-6432-4.* [Online] Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7249251 2017.01.02.

Dürkop, L., Jasperneite, J. & Fay, A., 2015. *An analysis of real-time ethernets with regard to their automatic configuration. In: Factory Communication Systems (WFCS).* IEEE World Conference , IEEE.

Egilmez, H. E., Dane, S. T. & Bagci, K. T., 2013. *OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks - IEEE. Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC).*

Gebauer, O., Ohms, J., Wermser, D. & Wähling, S.-O., 2016. *Mechanisms for the Automated Setup of Software-Defined Networks - ITG-Fachbericht 263 ISBN 978-38007-4220-2.* Osnabrück: VDE Verlag.

Handley, M., Jacobson, V. & Perkins, C., 2006. *RFC 4566 - SDP: Session Description Protocol.* [Online] Available from: https://tools.ietf.org/html/rfc4566 2017.01.05.

Herlich, M., 2016. Proof-of-concept for a software-defined real-time Ethernet. In: Emerging Technologies and Factory Automation (ETFA). *IEEE 21st International Conference* , pp. 1-4.

Jeong, K., Kim, J. & Kim, Y.-T., 2012. QoS-aware network operating system for software defined networking with generalized OpenFlows. *IEEE Network Operations and Management Symposium,* pp. 1167-1174.

Keqiang, H., 2015. *Latency in Software Defined Networks: Measurements and Mitigation Techniques - ACM 978-1-4503-3486-0/15/06.* [Online] Available from: https://aaron.gember-jacobson.com/docs/he2015sigmetrics.pdf 2017.01.08.

O´Doherty, P. & Ranganathan, M., 2017. *JSIP: Java SIP specification Reference Implementation.* [Online] Available from: https://github.com/usnistgov/jsip 2017.01.06

Open Networking Foundation, 2012. *OpenFlow Switch Specification.* [Online] Available from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf 2017.01.09.

Project Floodlight, 2017. *Floodlight OpenFlow Controller - Project Floodlight.* [Online] Available from: http://www.projectfloodlight.org/floodlight/ 2017.01.11.

Rosenberg, J., 2002. *RFC 2543 - SIP: Session Initiation Protocol.* [Online] Available from: https://www.ietf.org/rfc/rfc3261.txt 2017.01.04

Schulzrinne, H., Casner, S., Frederick, R. & Jacobson, V., 2003. *RFC 3550 - RTP: A Transport Protocol for Real-Time Applications.* [Online] Available from: https://tools.ietf.org/html/rfc3550 2016.12.15.

Thomas, R., 2000. *Architectural styles and the design of network-based software architectures - Doctoral Thesis.* University of California,: s.n.

Tsagkrais, K., 2015. Customizable autonomic network management: integrating autonomic network management and software-defined networking. *IEEE Vehicular Technology Magazine,* 10(1), pp. 61-68.

Wallner, R. & Cannistra, R., 2013. An SDN approach: quality of service using big switch's floodlight open-source controller. Proceedings of the Asia-Pacific Advanced Network. *Proceedings of the Asia-Pacific Advanced Network,* Issue 35, pp. 14-19.