Multimedia Optimized Routing in OpenFlow Networks

Michael Karl, *Student Member, IEEE*, Jochen Gruen, and Thorsten Herfet, *Senior Member, IEEE*
Telecommunications Lab
Saarland University, D-66111 Saarbruecken, Germany
Email: {karl,gruen,herfet}@nt.uni-saarland.de

***Abstract*—Networks are experiencing a paradigm shift from textual data to high data rate traffic, especially carrying streaming media. Traditional network architectures can generally not keep up with this change. Software Defined Network architectures try to enable network engineers to more easily develop new transmission architectures. Current approaches of media transport, e.g. as defined by RTSP/RTP, are purely end-to-end oriented and do not consider the architecture of the underlying network. In this work we propose ways of analyzing the network state and intercepting RTSP/RTP streaming packets in Software Defined Networks with the OpenFlow 1.0 protocol. This information is used for an adept routing decision. Finally, we share our view on sample experimental scenarios where we demonstrate the benefits of the proposed transmission architecture.**

Keywords: Software Defined Networking, OpenFlow, Streaming, RTSP/RTP

I. Introduction

Today, a large share of the Internet traffic is made up of video data. This video data comes in many different forms like Video on Demand (VoD), TV, P2P file sharing and video conferencing. It is expected that the total global share of video data in the traffic on the Internet will be around 86 percent by 2016 [1]. Therefore, dealing with the challenges of video transport over the Internet is an important topic to improve efficiency to cope with future requirements. Not only the QoS demands for video transmissions, e.g. in terms of data rate and delay, need to be met but also the resource utilization in the network should be optimized. Both requirements are not an inherent feature of the best-effort oriented IP network architecture in the Internet. Currently, Content Delivery Networks (CDN) are build to handle this arising challenge [7]. For example, Akamai is one of the global players and produces a lot of interest in the web sector. Another promising approach to deal with these challenges is Software Defined Networking (SDN). Thereby, the network is logically divided into two parts: the controlling and the forwarding plane. The controlling functionality is concentrated at a special entity, called controller. The forwarding plane consists of all network nodes that run the SDN implementation. With this separation, all nodes in the network are responsible for packet forwarding only. In contrast, the controller must manage all routing and maintenance information of the network. A popular approach to implement a software defined network is OpenFlow[1]. OpenFlow (OF) is a protocol specification for the communication between network nodes and the controller. With OpenFlow, packet transmissions are identified via *flows*. The controller can instruct the nodes to match incoming packets against predefined flows and associate an action with them, e.g. forwarding to a port or modify some fields of the packet header. This enables the controller to decide the routing and general packet handling of IP traffic through the network. These capabilities allow software defined networks, e.g. implemented by OpenFlow, to be more flexible than the traditional IP networks. Normal traffic can be handled as before with the usual IP routing mechanisms. But in case the need for special ways of traffic routing arises, this is possible by changing only one controller component in the network. SDNs can be useful for different applications that pose certain requirements, e.g. delay and data rate, towards the network. For example, these could be media streaming applications. Especially for campus networks between distributed company or university sites, SDN optimized routing is well applicable because the network architecture is controlled by a single entity. Google already uses an OpenFlow powered SDN solution for its internal network that handles datacenter-to-datacenter traffic to improve manageability, performance, utilization and cost-efficiency of their WAN [2]. Also, applications for remote collaboration [4] that heavily rely on video streaming can benefit from an improved routing technique of their media streams. Generally, SDNs can be utilized to route video traffic over special parts of the network that are well suited for this kind of traffic. Another possibility includes coordination of different traffic streams for a better exploitation of the available network resources. As IP networks usually only provide a best effort service, they can not guarantee the QoS requirements of media streams. Therefore, the OpenFlow standard presents a viable approach to allow a customized optimization of the transport based on the architecture of the underlying network.

In this paper we show how the messages of media session setup protocols like e.g. RTSP and SIP can be intercepted transparently by an SDN to retrieve information about the following out-of-band media transmission. This information can then be applied to improve the routing decisions for the media transmission. Due to space constraints we focus on the interception of RTSP traffic in OpenFlow networks.

The remainder of this paper is structured as follows: In section II we introduce the basic standards for H.264 video

[1] *http://www.opennetworking.org*

streaming via RTSP/RTP and state some requirements for media streaming applications. Afterwards, we describe the OpenFlow standard in section III. Section IV presents techniques to analyze an OpenFlow network topology. Eventually, we describe the interception of RTSP traffic and optimized forwarding mechanism proposed in this paper and potential traps during implementation in section V. Finally, section VI demonstrates some application scenarios and section VII concludes the paper.

II. RTSP/RTP AND H.264 FOR VIDEO CONFERENCING

The RTSP/RTP/RTCP suite [11][9] is part of the set of standards which provide specifications for multimedia transport. Together with the H.264/AVC video coding standard [10] and its encapsulation into the RTP protocol [8] they build a complete set of tools for video transmission pipelines. The RTSP standard defines the session setup and management and can be seen as a remote control for video streaming applications. It provides the ability to communicate implemented control functionality, setup sessions, and to describe and control the video streams, e.g. start, pause, that are available at the server. For the transmission of the actual video stream the RTP protocol is used. It basically defines a proper encapsulation of the video content into IP packets. Additional functionality includes measurement of the channel quality and synchronization of both video and audio streams via RTCP. Finally, the H.264 standard offers capabilities for error resilient, efficient, and flexible video coding of uncompressed video streams before transmitting over an RTP connection to the remote client.

Video streaming applications demand for multiple requirements that must be met to satisfy a decent user experience. These requirements can be described with the PRPD (Predictable Reliability under Predictable Delay) concept as introduced in [5]. Obviously, a balance between delivery delay and residual error rate is an essential part of an acceptable transmission. Depending on the scenario, a timely delivery is more desirable even if a small amount of residual errors remain. This is especially important for remote collaboration scenarios: for lip-synchronous audio/video streams the delay must be limited to at most $300ms$ while for video only streams it should not exceed $1000ms$. Delays that exceed these limits would severely degrade the task completion performance as shown in [6].

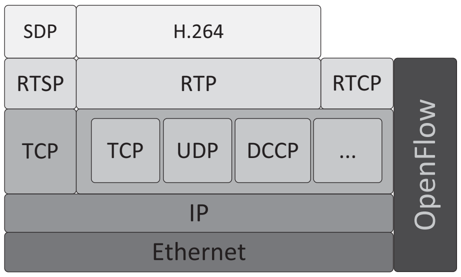

Fig. 1. Layered structure of the video streaming and OpenFlow standards similar to the OSI layer model.

Ethertype	MAC_SRC	MAC_DST	VLAN_ID	VLAN_PCP	
Proto_ID	IP_SRC	IP_DST	IP_TOS	Port_SRC	Port_DST

Fig. 2. Fields of a *flow* in OpenFlow 1.0

III. SOFTWARE-DEFINED NETWORKING

Software-defined networking (SDN) basically is a new approach to control networks with a separate controlling entity. The OpenFlow specification represents the probably most popular specification for the communication between controller and nodes in software-defined networks. In fact it defines a standardized communication and modification connection between controlling intelligence and the switching hardware. Basically, the architecture consists of two separated parts: controller and forwarding entities. Thus, the switching logic moves completely out of the forwarding hardware providing a central management and easy network configuration. In principle, with SDN and OpenFlow it is possible to create a spatially distributed switch with a central intelligence instance. The key strategy with OpenFlow is to match packets according to their *transmission signature*, called *flow*, and provide an individual way of forwarding through the network. A flow consists of data-link, network and transport layer characteristics, e.g. MAC/IP-addresses and communication ports. Figure 2 depicts the fields of a flow with OpenFlow 1.0. It is also possible to wildcard some flow fields to create more generic matches. When the flow is used to match a flow, it can be combined with a special action. For example, such an action can be a simple port forwarding command or a more complex MAC and IP field modification. This flexible handling of packets constitutes a clear advantage compared to the traditional static networking architecture where no differentiation between packets according to their transmission signature is made. This in turn enables flexible interactions with individually selected traffic. The SDN approach works as follows: Basically, a packet of a transmission reaches an edge OpenFlow node. In case the node has no information how to handle the packets, the default action is to send this packet via the OpenFlow protocol from this node to the controller. Then, the controller extracts all information from the packet and thus obtains its *flow* information. Obviously, all packets belonging to the same transmission have the same *flow* information. According to this information, the controller decides how to handle these packets. Therefore, it associates this *flow* with a *flow* signature and a set of special actions per node. The signature and the necessary actions are then stored at each network node that is required to forward this *flow*. The following packets of the transmission thus match the *flow* signature at the node. In this way, they can be associated with a set of actions at the nodes, e.g. port forwarding, which can be directly executed in the network without the packet being sent to the controller again. These entries can also be forced to timeout after a predefined time period. Eventually, software-defined networking provides a practical approach to overcome the limitations of current networks that were designed for traditional client-server communication. Furthermore, it prepares the way for the future media Internet with its high data rate and high quality applications. Obviously, innovations regarding new transport protocols and transportation systems are drastically eased.

IV. Network Analysis

Network statistics are a crucial part when optimizing resource utilization. The OpenFlow specification provides several raw node statistics as number of packets or bytes per link-port, but the evaluation of this information is left to the controller application. We periodically collect and store the port statistics of all available OpenFlow nodes in a centrally managed data structure Γ. In the following we describe how to calculate overall *packet loss rate*, *data rate* and *delay* of individual links in a network solely using the available command set of the OpenFlow 1.0 specification. Note that the approaches presented here are very basic. More sophisticated algorithms may help to improve precision. We assume that a directed link l is defined as a set of tuples (n, p) where n reflects the OF node and p denotes the physical connection port. Thus, we define a link l from node n_i to n_j as $l_{i,j} = ((n_i, p_i), (n_j, p_j))$. Additionally, to access the source or destination node and port of a link one uses $src(l_{i,j}) = (n_i, p_i)$ or $dst(l_{i,j}) = (n_j, p_j)$, respectively.

1) Packet Loss Rate: To ensure a smooth packet loss rate (PLR) estimation a number of samples N_{sample} for transmitted and received packets $N_{tx}(n, p)$ and $N_{rx}(n, p)$ at node n and connection port p is stored. The mean over all values is calculated as

$$N_{tx|rx,mean}(n,p) = \frac{\sum_{k=1}^{N_{sample}} N^k_{tx|rx}(n,p)}{N_{sample}}$$

Thus, the packet loss rate on a link l between two nodes n_i and n_j can be obtained by

$$PLR(l) = \frac{N_{tx,mean}(src(l)) - N_{rx,mean}(dst(l))}{N_{tx,mean}(src(l))}$$

2) Consumed Data Rate: The consumed data rate $D(l)$ on a link l is the number of bytes that is transmitted. The calculation resembles the packet loss calculation. The number of transmitted and received bytes, that is $B_{tx}(n, p)$ and $B_{rx}(n, p)$, at a node n and port p is extracted from Γ at two subsequent and well-defined time instances t_1 and $t_2 > t_1$. Thus, the transmit and receive data rate at this node can be calculated as

$$D_{tx}(l) = \frac{B_{tx,2}(src(l)) - B_{tx,1}(src(l))}{t_2 - t_1}$$
$$D_{rx}(l) = \frac{B_{rx,2}(dst(l)) - B_{rx,1}(dst(l))}{t_2 - t_1}$$

3) Maximum Data Rate: The maximum data rate $\widehat{D}(l)$ that can be transmitted over a link l is used to compute the available data rate $D_{avail}(l)$:

$$D_{avail}(l) = \widehat{D}(l) - D_{tx}(l)$$

```
C->S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/1.0
      CSeq: 302
      Transport: RTP/AVP;unicast;destination=192.168.0.1;
      client_port=4588-4589

S->C: RTSP/1.0 200 OK
      CSeq: 302
      Date: 23 Jan 1997 15:35:06 GMT
      Session: 47112344
      Transport: RTP/AVP;unicast;destination=192.168.0.1;
      client_port=4588-4589;source=134.64.0.1;
      server_port=6256-6257
```

Fig. 3. Exemplary RTSP setup handshake (taken from [11]).

Here it is assumed that the maximum data rate $\widehat{D}(l)$ of each link is known to the controller.

4) One-Way Delay: Calculating the one-way delay is slightly more challenging since it is not possible to extract this network property directly from the OpenFlow specific command set. Therefore, we propose an indirect delay measurement for links. The controller creates a standard UDP packet which carries a timestamp t_0 as payload. This UDP packet is directly sent to one of the nodes connected to the desired link. This nodes forwards the UDP packet to the connected node which relays it back to the controller. There the packet's UDP payload is inspected and the timestamp extracted. Additionally, the packet's arrival time t_1 is stored. Thus, the one-way delay of the link can easily derived by $delay = t_1 - t_0$. Note, that this delay also includes the OpenFlow queuing delays at both nodes as well as the requested link's propagation delay. We assume a physically separate management network that introduces no further delay between controller and the nodes. Obviously, confusions between productive traffic and the measurement packets must be avoided. To achieve this, the OSI layer 2 and layer 3 characteristics of the UDP packet used for measurement must be specialized and unique to be able to match them at the receiving node and send it back to the controller. Otherwise productive traffic would also be matched and thus both, the measurement and the normal transmissions are affected. For example, this can be realized by specialized MAC addresses.

V. Optimizing RTSP/RTP traffic

This section describes how RTSP traffic can be intercepted and analyzed in OpenFlow environments. The knowledge about ongoing video sessions enables the optimization of routing decisions that are made for the RTP traffic that is following.

A. RTSP Interception

As OpenFlow is centrally coordinated by a controller instance, it is required that this entity obtains information about all video streams in the network. By making this device aware of the ongoing video sessions, it is able to optimize the process of routing the corresponding video streams through the network. Therefore, the controller has to inspect all RTSP messages that are exchanged between a server and any client. This assures that a session memory similar to that of the video streaming server can be established. To achieve this

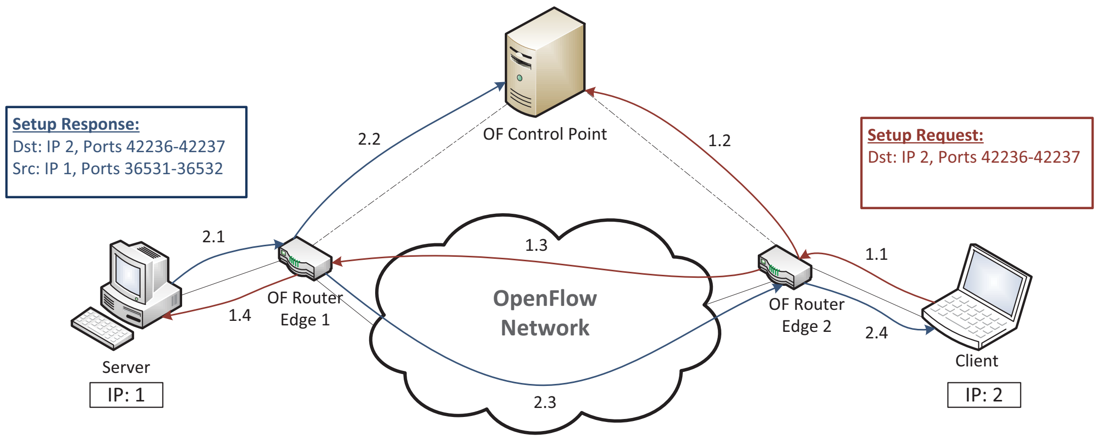

Fig. 4. The interception of an RTSP SETUP exchange by the controller in an OpenFlow network. Both RTSP messages are redirected to the OpenFlow Controller before forwarding them to the original destination. The setup response includes all IP addresses and ports that are assigned to the RTP transport connection.

the OpenFlow nodes at the borders of the network, which are closest to client and server, can be configured to forward any incoming RTSP packets to the controller. Currently, any packets with the default RTSP destination port 554 or 8554 are checked for valid RTSP content. In that way, the controller is able to parse the RTSP message stream and create the session memory mentioned above. Afterwards, the OpenFlow nodes forward the RTSP message packets to the correct destination in the network, that is the server/client. The most important message pair that the controller needs to parse is the RTSP `SETUP` handshake. The response of this handshake contains the `Session` and `Transport` fields that the server returns to the client. These fields contain amongst others the session id, the server/client IPs and ports, and the underlying transport protocol for the RTP connection over which the actual media transmission will take place. An exemplary `SETUP` handshake is shown in figure 3. With this knowledge it is possible to setup an OF flow that exactly matches the RTP transmission. The flow signature contains the client/server IPs and target ports as well as an identifier for the transmission protocol. The rest of the criteria in the flow signature are wildcarded. This includes also the source ports of the connections due to the lack of knowledge which port the application will choose. Figure 4 shows a sample scenario in which a server with IP 1 and a client with IP 2 are connected to the border nodes of an OpenFlow network. As soon as the client sends an RTSP `SETUP` request, the node forwards it to the controller. It in turn inspects the packet to update its video session memory. The same holds true for the response of the server to this request. Additionally, the figure illustrates that the packets contain the IPs and ports that are used in the out-of-band RTP connection for the audio/video transmission. A similar scheme can be applied to all other RTSP messages in case additional information is required. Note that this setup is completely transparent to existing software that implements the RTSP/RTP standards and does not require any change. This is advantageous when applying the optimization described in this work to already established applications.

B. *Optimized RTP Routing*

The main goal of this optimized forwarding mechanism is to cleverly utilize the present network resources. Software-defined networks better offer the ability to modify the transmission behavior than any other current architecture. Therefore, the optimum route for the RTP transmission according to the present network information can be calculated. Our procedure is based on traditional shortest-path algorithms as described in [12]. Note, that we provide a very basic algorithmic solution on purpose. Analyzing and optimizing the routing decisions can be part of future research. Principally, it is possible to extend the search and combine multiple criteria to use them in legacy graph algorithms as shown in [3]. In this work we assume either the minimum number of hops, the round-trip time (RTT), packet-loss rate (PLR) or data rate of links as the single routing criterion. The last step is to install the flow in the network that is described by the flow signature described in section V-A. The implementation of the proposed mechanism presumes a set of controller features to collect and process the required information. Thus, multiple implementation aspects must be considered. A good practice is to set up the flow forwarding route starting from the node closest to the receiver of the stream. This avoids glitches during route setup due to the asynchronous nature of the network. The most critical aspect with the controller design is to *handle high data rate streams* when the flows are setup on-demand and not a priori by the administrator. It turned out that the NOX 0.9.1 controller implementation does not perform well with a bunch of packets arriving at the same time. This is valid for both regular data streams and all by the controller itself initiated sequences of packets, e.g. required for the delay measurement. Thus, we propose to store the flow-signature at the controller for a specific time and skip all further packets with this signature until the entry times out. Otherwise, the controller performance is highly degraded due to the individual processing of each packet as long as the flow has not been set up. The *periodical polling* feature which is required to poll the gathered information from the nodes of the network is a highly critical part that reflects the accuracy of the final statistics. Obviously, more frequent polling leads to a more accurate topology view but introduces more traffic and computation. In addition, the polling-request for statistics from nodes must be coordinated to avoid packet congestion at the controller. To enable a rating of routes for an emerging stream in case of e.g. multi-stream scenarios a forwarding-route *bookkeeping* feature is introduced. It allows to compare possible routes not only on the information regarding the network topology but also on the routes that were established for already existing flows. With this features it is also possible to update existing routes to the most current state of the network topology by performing periodical tests. All this information must be stored in an efficient way, that is accessing these information must be done quickly to not degrade the controller performance. Especially in large networks, where the set of statistical information of all connected nodes may grow rapidly, a fast access is essential. Eventually, the number and character of controller features highly influences the required complexity and may be aligned to the present hardware.

VI. Application Scenario

The testbed consists of six OpenFlow nodes and one NOX 0.9.1 controller. The nodes are running OpenFlow 1.0 on generic Ubuntu 10.4.3 machines. Both sender and receiver are also regular Ubuntu 10.4.3 machines. The OpenFlow nodes are out-of-band connected to the controller via a private class C network. The testbed topology is illustrated in figure 5. More information about the testbed can be found at our OpenFlow project website[2]. We assume that the OpenFlow network properties round-trip time, packet loss rate and maximum link data rate are fixed and, additionally, there are neither node nor link errors. The connections to sender and receiver are considered as optimal. The link properties are presented in table I. The values are due to the effect that we create an overlay network on top of a productive network in which there is already existing traffic. Therefore, we see the presented data rate as the remaining available data rate for the overlay links. Similarly, the packet loss rates are due to congestion and the heterogeneous structure of the overlay links which consist of multiple different physical links.

In the following we present a set of test scenarios that show the benefits of our approach. In all test scenarios we apply the Dijkstra algorithm although with a different metric to determine the edge weights. The result of each scenario is benchmarked against the results of a Dijkstra with min-hop metric in the same scenario.

A. Multiple Video Streams

In this scenario both receivers request independent video streams of roughly 1Mbps. In case of minimum hop routing,

[2]*http://www.openflow.uni-saarland.de*

TABLE I
OpenFlow Link Characteristics

Link	RTT $[ms]$	PLR	Data rate[Mbps]
Node 1 - Node 2	9	0.02	1.5
Node 1 - Node 3	8	0.03	1.5
Node 1 - Node 4	10	0.01	1.5
Node 2 - Node 6	12	0.01	1.5
Node 3 - Node 4	7	0.01	1.5
Node 3 - Node 5	11	0.02	1.5
Node 4 - Node 5	5	0.01	1.5
Node 5 - Node 6	3	0.02	1.5

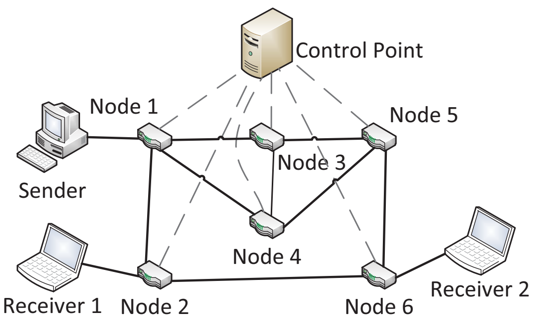

Fig. 5. The setup of our testbed. It includes five OpenFlow switches and one controller.

the setup of the testbed causes both video streams to be routed via node 2. As the sum of the data rates of both video streams exceeds the available data rate on that link, packets of both streams will be dropped. This leads to artifacts in the video presentation at both receivers. Depending on the transmission protocol either additional delays (TCP) or image artifacts (UDP) due to lost packets occur. To avoid this, we iteratively apply the Dijkstra algorithm with special link weights. For each stream that needs to be routed through the network the edge weights are recalculated incorporating the already routed streams. Additionally, the Dijkstra algorithm is applied for each stream to find the optimal path for it. Assume there are $k \geq 1$ streams, each with a data rate $D_{stream,k}$. In case there are already $i \geq 0$ streams in the network, we achieve the currently available data rate on a link l as

$$\bar{D}_i(l) = \begin{cases} D_{avail}(l) & , i = 0 \\ \bar{D}_{i-1}(l) - D_{stream,i} & , i \geq 1 \end{cases}.$$

The maximum available data rate in the network with i streams is calculated as $\bar{D}_{max,i} = \max_l(\bar{D}_i(l))$. Then, the Dijkstra link weights $W_i(l)$ for a network with $i \geq 0$ already established streams are

$$W_i(l) = \begin{cases} w(\bar{D}_{max,i} - \bar{D}_i(l)) + 1 & , \bar{D}_i(l) \geq D_{stream,i+1} \\ 2 \cdot w(\bar{D}_{max,i} - \bar{D}_i(l)) + 1 & , \text{otherwise} \end{cases}.$$

where $w(\cdot)$ is a function that maps data rate D with $[D] = kbps$ to $\mathbb{N}$. It can be defined as:

$$w(D) = D \cdot \frac{s}{kbit}.$$

Therefore, the flows that are finally installed in the OpenFlow network depend on the order in which the streams are added

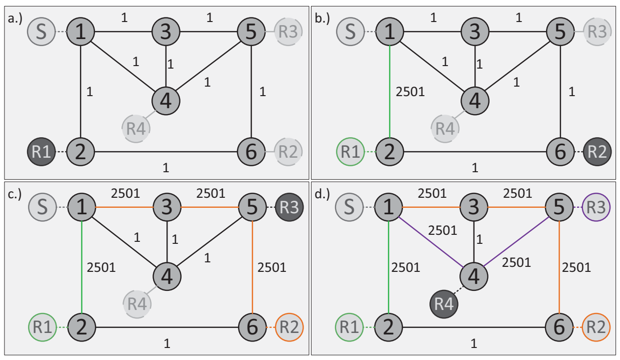

Fig. 6. An example for the multi-stream scenario. It shows the gradual change of link weights and therefore the modified routing decisions.

into the network. Figure 6 illustrates an example for the proposed data rate routing algorithm. It is still the case that all receivers request a stream with 1Mbps. Part a) of the figure shows the network with no ($i = 0$) streams currently established in the network. Hence, all link weights are $W_0(l) = 1$. Parts b)-d) show the link weights when the receiver $R1 - R3$ are being gradually connected to the sender. In this case the route to receiver $R1$ is computed first and we end up with a route via nodes $1, 2$. Clearly, the corresponding link weight increases to 2501 since the available data rate on this link is smaller than the stream data rate. When receiver $R2$ connects to the sender, the data is routed via nodes $1, 3, 5, 6$ since the link between node 1 and node 2 is already saturated in the streaming direction. Next, receiver $R3$ is served with data via nodes $1, 4, 5$ since all other paths can not provide a sufficient available data rate. Finally, receiver $R4$ must be connected via nodes $1, 4$. This represents the case where the network is out of resources and the application must adapt to the missing transmission capacity.

For example, consider a scenario where a sender transmits a stream via UDP. In this case it is crucial to determine a network route that has a low end-to-end packet loss rate PLR_{e2e} as UDP does not implement any error-correction techniques. Hence, transmission reliability has a higher priority than transmission time since the receiving client can not correct any disturbed or request any lost packets. Streaming data via UDP from the sender to receiver 2 would therefore be delivered via nodes 1,4,5,6 with $RTT_{e2e} = 21ms$ and $PLR_{e2e} = 1 - (1 - 0.01)^2 \cdot (1 - 0.02) = 0.0298$.

VII. Conclusion

This paper proposes a streaming setup that utilizes the flexible routing possibilities provided by an SDN implemented with OpenFlow. Thereby, the quality of the transmission channel for RTSP/RTP streaming applications is transparently improved, that means no interaction from the communication endpoints is required. We showed how to learn network analytics and intercept RTSP/RTP streaming packets in SDNs to exploit these information for a skillful routing decision. Additionally, multiple implementation aspects are clarified to demonstrate potential traps. Finally, we provide a basic experimental scenarios that shows the promising benefit for network resource utilization when using an SDN/OpenFlow routing approach.

Acknowledgment

The work presented in this paper was performed in the context of the Software-Cluster projects EMERGENT and SINNODIUM (www.software-cluster.org). It was partially funded by the German Federal Ministry of Education and Research (BMBF) under grant no. ”01IC10S01” and ”01IC12S01”. The authors assume responsibility for the content.

References

[1] Cisco Systems, Visual Networking Index, *Entering the Zettabyte Era*, January 2013

[2] Google, ”Inter-Datacenter WAN with centralized TE using SDN and OpenFlow”, January 2013

[3] Karl, M.; Polishchuk, T.; Herfet, Th.; Gurtov, A., ”Mediating Multimedia Traffic With Strict Delivery Constraints”, IEEE International Symposium on Multimedia (ISM), December 2012, Irvine, USA.

[4] Gruen, J.; Gerber, T.; Herfet, Th., ”Multi-Reality Interfaces - Remote Monitoring and Maintenance in modular Factory Environments”, IFAC Symposium on Information Control Problems in Manufacturing (INCOM-2012), May 2012, Bucharest, Romania.

[5] Gorius, M.; Miroll, J.; Karl, M.; Herfet, Th., *Predictable Reliability and Packet Loss Domain Separation for IP Media Delivery*, IEEE International Conference on Communications (ICC) 2011, June 2011

[6] Gergle, D.; Kraut, Robert E.; Fussell, Susan R., ”The Impact of Delayed Visual Feedback on Collaborative Performance”, CHI 2006 Proceedings, April 2006, Qubec, Canada.

[7] Pallis G.; Vakali, A., *Insight and Perspectives for Content Delivery Networks*, Communications of the ACM, Vol. 49, No. 1, 2006

[8] Wenger, S.; Hannuksela, M.; Stockhammer, T.; Westerlund, M., and D. Singer, ”RTP Payload Format for H.264 Video”, RFC 3984, February 2005.

[9] Schulzrinne, H.; Casner, S.; Frederick, R.; V. Jacobson, ”RTP: A Transport Protocol for Real-Time Applications”, STD 64, RFC 3550, July 2003.

[10] Wiegand, T.; Sullivan, G. J.; Bjontegaard, G.; Luthra, A., ”Overview of the H.264/AVC video coding standard”, IEEE Transactions on Circuits and Systems for Video Technology In Circuits and Systems for Video Technology, IEEE Transactions on, Vol. 13, No. 7., 04 July 2003, pp. 560-576.

[11] Schulzrinne, H.; Rao, A.; and R. Lanphier, ”Real Time Streaming Protocol (RTSP)”, RFC 2326, April 1998.

[12] Cormen, Th.; Leiserson, Ch.; Rivest, R.; Stein, C., *Introduction to Algorithms*, 3rd Edition, MIT Press, p. 643ff