2014 IEEE 28th International Conference on Advanced Information Networking and Applications

Transparent Multi-Hop Protocol Termination

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

***Abstract*—According to Cisco's forecast by the end of 2016 the global annual IP traffic will surpass 1.3 zettabytes and mobile traffic will reach a volume of 11.2 exabytes per month in 2017. Thus, a main requirement for future networks is high reliability and efficiency to cope with the associated quality of service demands. Hence, a vital interaction between error-correction, network technologies and suitable network resource utilization constitute a hard challenge for traffic engineers. In prior work it has been shown that the data distribution with traditional end-to-end schemes is less optimal compared to multi-hop schemes, that is treating network segments individually. In this paper we present a theoretical analysis based on Shannon's coding theorem that concludes a promising data-rate reduction from the segmentation approach. Additionally, we propose a transparent protocol termination mechanism for software-defined networks (SDN) that allows to segment an end-to-end transmission path without any modification at the sending or receiving devices. Eventually, a detailed implementation approach for SDNs running OpenFlow is given.**

Keywords: Congestion Avoidance, Multi-Hop Transmission, Software Defined Networking, OpenFlow

I. Introduction

According to a Cisco forecast [1], by the end of 2016 global IP traffic will surpass the 110 exabytes per month in the Internet which is characterized by a highly heterogeneous and complex topology. For example, the Internet consists of various autonomous systems (AS), where each AS is individually managed and maintained by a network operator. Also, current network structures, especially the Internet, become more dynamic and mission critical since they provide the basis for real-time services. One popular example is mobile video. Cisco's VNI [1] predicts that approximately 66% of the global traffic is requested by mobile video applications. Due to its special mobile character it has even more strict requirements to quality of service (QoS) and the resulting user experience. Therefore, an efficient and customized traffic management is essential. A general approach to achieve this is *traffic engineering* which focuses on the performance optimization of operational IP networks. This in turn leads to a clever distribution of required data-rates for all transmissions in the networks and may lower transport delays. Furthermore, error-correction is a highly important topic. Despite the fact that the majority of lost packets in large networks are due to congestion, particularly wireless connections also introduce significant packet losses due to varying channel conditions. Commonly, radio based network links are used for the *last mile* segment towards mobile devices. Thus, these last mile connections are primarily responsible for packet losses due to packet corruptions. When considering mobile video applications this obviously constitutes the quality bottleneck. Correcting errors in networks assumes that there is some extra data besides the raw data transmission to cope with potential losses during transmission. This redundant data is called *redundancy information (RI)*. Traditionally, the amount of redundancy is calculated based on the transmission properties between the communication end points, the sender and receiver. Obviously, the data-rate increases according to the required redundancy. In general, the data packets pass multiple different links in different AS, each with individual characteristics. Thus, the amount of redundancy added to the original data is not optimal, since the calculation is based on a virtual connection (end-to-end). The properties of this end-to-end connection are a merged version that mask many attributes of the underlying links. A possible way to uncover these normally blurred information is to split an end-to-end transmission into smaller parts. Thus, the transport protocol must be terminated somewhere in the center of the network, not only at its edges. The protocol termination must be transparent to the sending and receiving application in order to avoid changes to these applications. Traditional IP networking approaches do not provide an easy handling for sophisticated packet modification on the fly without involving the device vendors. The advent of software-defined networks (SDNs) implicates a set of new features for both, the development of new networking approaches and improvement of current networking mechanisms. As an example Google uses this new networking technology for their inter-data-center WAN [2]. In RFC 3135 *Performance Enhancing Proxies (PEPs)* are described which are deployed to improve degraded TCP performance caused by characteristics of specific link environments [7]. One operation mode of the proposed PEPs is *split connection*, which terminates a connection received from one end-device and establishes a new connection to the other end-device. The work presented in this paper is highly related to the described PEPs.

This paper extends this set of challenges by a *spatial networking dimension* and also proposes a corresponding solution. In this paper we use the channel coding theorem to analytically proof that segmenting a transmission path always result in less required data-rate per communication segment. Therefore, we base the calculations on Shannon's coding theorem [3] and assume no time constraints for coding. We show that a split transmission always outperforms the traditional end-to-end approach: the required data-rate with a split connection is always lower compared to the traditional

1550-445X/14 $31.00 © 2014 IEEE
DOI 10.1109/AINA.2014.35

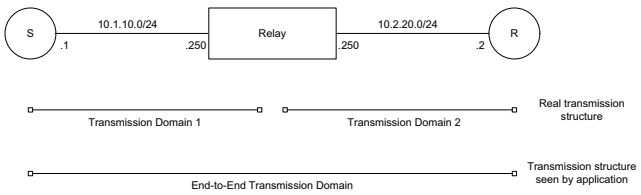

Fig. 1. Basic Relaying Mechanism.

end-to-end approach. Note, that this does not contradict prior work [5] which includes restricting the time allowed for error correction on each segment. Thus, splitting a connection into individual parts leads to a remarkable reduction of network utilization. Additionally, we compare both approaches and benchmark their redundancy performance in a two link scenario. Furthermore, a detailed practical approach of transparent protocol termination for OpenFlow based software-defined networks (SDN) [4] is proposed.

The paper is structured as follows: Section II provides a general overview on the problem focused in this paper. Section III provides an analysis of end-to-end and split transmission approaches based on the channel coding theorem. Section IV briefly introduces software-defined networks. Section V provides a full approach for SDNs to implement a transparent protocol termination with relays for both, uni- and bi-directional protocols. Section VI evaluates the proposed mechanism based on a TCP transmission. Section VII concludes the paper.

II. Problem Statement

The global IP traffic travels along multiple autonomous systems and crosses a large set of varying links. Generally, links can be coarsely categorized into two groups: wired and wireless. Wired links often lose packets due to congestion issues whereas wireless connections are prevalently responsible for real packet distortions due to radio signal perturbations. Commonly, an Internet connection is established over multiple different transmission links which of course includes wired and wireless connections. Interestingly, the Cisco VNI also forecasts that the traffic generated by wireless devices will surpass traffic initiated by wired networking devices. Thereby, the demand for reliable and efficient error-correction schemes is growing. Current correction schemes focus on the traditional end-to-end connection control approach where the whole network part between source and sink is abstracted to one virtual link. Hence, the total spatial dimension of the network is peculated and thus simply lost. Prior work [5] [6] shows that exploiting this spatial dimension opens a promising opportunity for error-correction approaches to further increase efficiency especially if the transmission time is strictly limited. This in turn leads to reduction of the required transmission data-rate over error-prone networks due to less redundancy data. In this paper we focus on the reduction of the average per-link data-rate based on individual error-correction by segmenting the network in case the maximum allowed end-to-end transmission time is infinite. Thus, this paper directly relies on the application of Shannon's coding theorem [3].

III. Theoretical Analysis

For the theoretical analysis we assume the communication to be modeled by an erasure channel, that is either a packet is received without any error or it is completely lost. In the context of this paper, packet losses occur only due to real disturbances, e.g. in wireless LANs, and not due to congestion. We focus on a theoretical case which relies on the *Noisy-Channel Coding Theorem* [3] and does not assume a specific error-correction scheme or approach. In the following, a *link* always refers to either a real physical link or to a virtual segment, containing multiple physical links.

A. Coding and Redundancy

In summary, Shannon's noisy channel coding theorem states that as soon as an error-coding technique has infinite time, each arbitrary low probability of error during a transmission can be achieved. This can intuitively justified when considering an ARQ retransmission scheme: if the scheme has infinite time, it can gradually send retransmission packets until all packets have been received correctly, no matter how long it takes. The theoretical minimum required redundancy for a link with loss probability p is defined as $RI_{min} = \frac{p}{1-p}$. This value holds true for any error-coding scheme in case there is infinite time for error-handling. Obviously, practical RI values are higher than RI_{min}. With that, we define the data-rate after error-correction coding as $D = D_0 \cdot (1 + RI)$, where D_0 denotes the original data-rate of the source and the required RI depends on the used correction-scheme. Furthermore, we define $\lambda = \frac{D}{D_0} \geq 1$ as the *redundancy overhead*. It relates the coded streaming data-rate to the uncoded. If $\lambda = 1$, all links have zero probability of loss and thus no redundancy is required and $D = D_0$. In case $\lambda > 1$ redundancy is added since at least one $p_i > 0$ and thus $D > D_0$. In the following, we use λ to compare the data-rates for two different scenarios: *end-to-end* and *split*.

B. End-to-End Transmission

In case we consider an end-to-end scenario, the overall connection is modeled as a single virtual link. Let us assume that there are n subsequent links on the path between sender and receiver. Each link i has loss probability p_i, and thus the end-to-end loss probability is $p_{e2e} = 1 - \prod_{i=1}^{n}(1 - p_i)$. For the average per link data-rate in a traditional error-correction end-to-end application scheme, it holds

$$\begin{aligned} D_{e2e} &= (1 + RI_{e2e}) \cdot D_0 \\ &= \left(1 + \frac{1 - \prod_{i=1}^{n}(1-p_i)}{\prod_{i=1}^{n}(1-p_i)}\right) \cdot D_0 \\ &= D_0 \cdot \underbrace{\frac{1}{\prod_{i=1}^{n}(1-p_i)}}_{:=\lambda_{e2e}} \end{aligned}$$

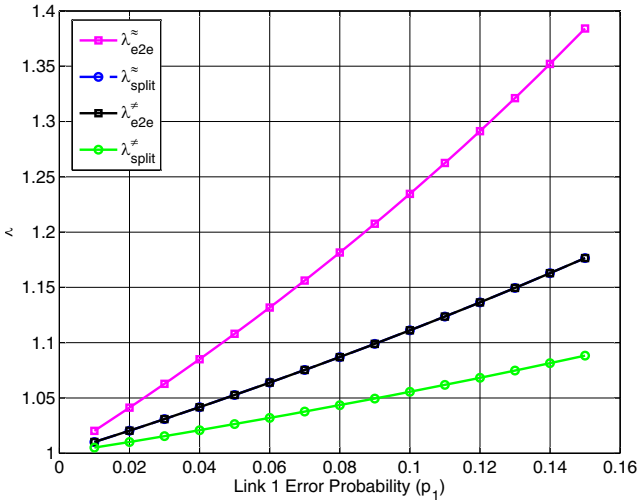

Fig. 2. Comparison of the overhead factors λ. Note that $\lambda^{\approx}_{e2e} = \lambda^{\neq}_{split}$ and thus the lines are equal.

C. Split Transmission

Next, we consider the case where the transmission is split among n path segments, where each segment i has loss probability p_i. It holds

$$\begin{aligned} D_{split} &= \frac{\sum_{i=1}^{n}(1+RI_i)}{n}\cdot D_0 = \frac{D_0}{n}\cdot\sum_{i=1}^{n}\left(\frac{1}{1-p_i}\right) \\ &= \frac{D_0}{n}\cdot\frac{\sum_{i=1}^{n}\left[\prod_{j=1,j\neq i}^{n}(1-p_i)\cdot(1-p_j)\right]}{\prod_{i=1}^{n}(1-p_i)} \\ &= D_0\cdot\underbrace{\frac{\sum_{i=1}^{n}\left[\prod_{j=1,j\neq i}^{n}(1-p_i)\cdot(1-p_j)\right]}{n\cdot\prod_{i=1}^{n}(1-p_i)}}_{=:\lambda_{split}} \end{aligned}$$

D. Two Link Scenario

A simple application scenario for a split error-correction application is with two subsequent network segments. Now, we discuss the case of a two link path where p_1 is arbitrary but fixed and both loss probabilities are a) nearly equal ($p_1 \approx p_2$) together with b) highly different ($p_1 >> p_2$). Due to the structures of D_{e2e} and D_{split} it is sufficient to limit our focus on the overhead factors λ_{e2e} and λ_{split}. For both cases we provide an efficiency factor $\Lambda = \frac{\lambda_{split}}{\lambda_{e2e}}$ that reflects the benefit of using a split approach compared to an end-to-end approach in terms of reduced data-rate.

First, we consider the case where $p_1 \approx p_2$. Starting the analysis with an end-to-end approach one obtains

$$\lambda^{\approx}_{e2e} = \frac{1}{(1-p_1)\cdot(1-p_2)} \approx \frac{1}{p_1^2 - 2\cdot p_1 + 1}$$

Differentiation of this term delivers its slope $\frac{\partial}{\partial p_1}\lambda^{\approx}_{e2e} = \frac{2}{(1-p_1)^3}$. Figure 2 shows the evaluation of $\lambda^{\approx}_{e2e}$ for the multiple values of p_1. In contrast to this, the split application leads to

$$\begin{aligned} \lambda^{\approx}_{split} &= \frac{(1-p_1)+(1-p_2)}{2\cdot(1-p_1)\cdot(1-p_2)} \\ &\approx \frac{2-2\cdot p_1}{2\cdot(1-p_1)^2} = \frac{1}{1-p_1} \end{aligned}$$

Figure 2 also illustrates the evaluation of $\lambda^{\approx}_{split}$. One easily identifies $\lambda_{e2e} \geq \lambda_{split}$ and notices that the split transmission approach leads to a significant total reduction of required redundancy. For example, if both links have an error-probability of $p_1 = p_2 = 0.07$, the total difference is 0.09. In case $D_0 = 1Mbit/s$, the split approach would save $0.09Mbit/s$ per link on average for one transmission. Obviously, the saving scales linearly with the number of transmissions on these links. The data-rate reduction factor between end-to-end and split application for nearly equal loss probabilities is

$$\Lambda^{\approx} = \frac{\lambda_{split}}{\lambda_{e2e}} = \frac{(1-p_1)^2}{1-p_1} = 1-p_1$$

Analyzing the structure of $\Lambda^{\approx}$ shows that the higher the link error-probabilities, the more data-rate can be saved in case a split transmission approach is used. Now, let us focus on the scenario where $p_1 >> p_2$ and thus one link dominates the other in terms of error-probability. With an end-to-end approach one obtains

$$\begin{aligned} \lambda^{\neq}_{e2e} &= \frac{1}{(1-p_1)\cdot(1-p_2)} = \frac{1}{1-p_1-p_2+p_1\cdot p_2} \\ &\approx \frac{1}{1-2\cdot p_1+p_1^2} \\ &\approx \frac{1}{1-p_1} \end{aligned}$$

whereas the split application leads to

$$\begin{aligned} \lambda^{\neq}_{split} &= \frac{(1-p_1)+(1-p_2)}{2\cdot\prod_{i=1}^{2}(1-p_i)} \\ &= \frac{2-p_1-p_2}{2\cdot(1-p_1-p_2+p_1\cdot p_2)} \\ &\approx \frac{2-p_1}{2\cdot(1-p_1)} \\ &\approx \frac{1-\frac{p_1}{2}}{1-p_1} \end{aligned}$$

Figure 2 additionally illustrates the evaluation of $\lambda^{\neq}_{e2e}$ and $\lambda^{\neq}_{split}$. Again, it clearly holds $\lambda_{e2e} \geq \lambda_{split}$ and with the assumption that $p_1 = 0.07 >> p_2$ and $D_0 = 1MBit/s$ the split transmission leads a reduction of roughly $0.03MBit/s$ per link on average. Again, this data-rate reduction scales linearly with the number of transmissions. In this case, the data reduction factor between end-to-end and split application is

$$\Lambda^{\neq} = \frac{\lambda_{split}}{\lambda_{e2e}} = \frac{(1-p_1)\cdot(1-\frac{p_1}{2})}{1-p_1} = 1-\frac{p_1}{2}$$

Ultimately, if $p_1 \approx p_2$ the application of a split error-correction approach leads to a data-rate reduction of a factor $1-p$ per link on average, in case both links have nearly the same loss

Ethertype	MAC_SRC	MAC_DST	VLAN_ID	VLAN_PCP
Proto_ID	IP_SRC	IP_DST	IP_TOS	Port_SRC	Port_DST

Fig. 3. Fields of a *flow* in OpenFlow 1.0

probabilities. If $p_1 >> p_2$, the data-rate reduction per link on average is $1 - \frac{p_1}{2}$.

IV. Software-defined Networking

Software-defined networking (SDN) is an approach to control networks with a separate and central controlling entity. The OpenFlow specification[1] 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 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 3 depicts the characteristic fields of a flow signature with OpenFlow 1.0. It is also possible to wildcard some flow fields to create more generic matches. When the flow signature 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, best effort, 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: A packet of a transmission reaches an OpenFlow-enabled 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, due to its flexible and modular structure it prepares the way for the future media Internet with its high data-rate and high quality applications. Ultimately, innovations regarding new transport protocols and transportation systems are drastically eased.

[1]https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

V. Practical Implementation

In this section we propose a practical implementation of a transparent protocol termination in software-defined networks. We present a stream interception and relaying mechanism that works for both, unidirectional (e.g. UDP) and bidirectional (e.g. TCP) connections. In general, the OpenFlow network nodes are simple and potential closed-source network devices, but able to execute commands and store information sent by the SDN controller. Thus, it can be assumed that no further software is available on the OpenFlow node itself. This especially holds for CPU intensive applications as transport termination and error-correction operations. Therefore, the protocol termination can not be performed on this node, but on a node close to this. Optimally, the external device that is responsible for the termination is directly attached to the corresponding OpenFlow node in order to keep the delay low. Due to this reason, this OpenFlow node must match multiple flows and perform individual forwarding and modification actions. To make the relay independent of present network specifics such as network addresses and netmasks, it always assumes to get its input from a virtual sender and sends it relayed output to a virtual receiver. These virtual addresses must be carefully chosen to avoid overlaps with real existing addresses. Furthermore, note that a successful layer 3 addressing assumes a valid layer 2 addressing, that is, if the relay sends data to a virtual, non-existing address in a subnet, it initially tries to get its MAC address by using ARP request. Unfortunately, the request is never be answered since there is no physical entity that could answer. Thus, it is absolutely necessary that the relay has a static ARP-table entry that maps the virtual IP address to an arbitrary and unused MAC address (like DE:AD:CA:B1:E0:01) in order to omit ARP requests and thus a stalling relay process. In the following a mechanism is presented that is intended to work with all kinds of SDN as long as the packet headers up to OSI layer 4 can be modified. We only present the basic proceeding for one stream, in case multiple streams should be transparently terminated at the same time, an elaborated bookkeeping system is required.

A. Matching and Relaying Process

In this section we describe the mechanism of matching data streams and redirect them to the desired destination. Additionally, the controller has the information about stream

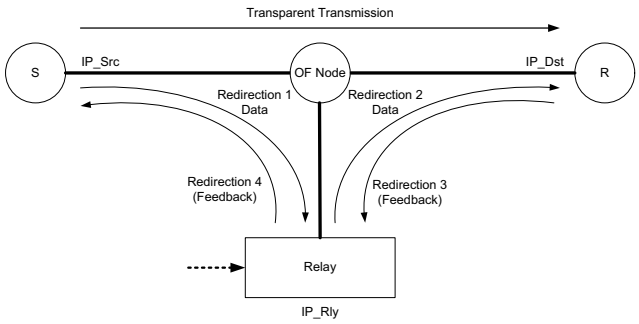

Fig. 4. Detailed OpenFlow based Redirection Scheme

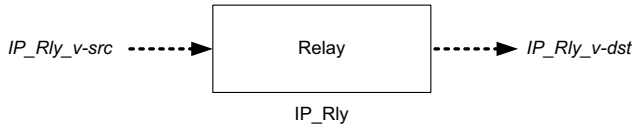

Fig. 5. Basic Relay Structure

types which can be handled by a relay and where the relay is connected. Thus, as soon as a packet arrives at the controller, it can be verified if there is a relay on the network path and if this relay is able to handle the stream. Generally, this can be done by identifying the *transport destination port*. We only describe the process of transparent relaying at one node. It is also possible to extend this mechanism to multiple relays on a path. Then, obviously each node must perform these actions and the overall end-to-end transmission is split into multiple segments, each with its own transmission and error-correction control. Eventually, the sender and receiver do not recognize any changes during transmission. Note, that during the matching the sending source ports are always wildcarded since it is principally not possible to determine them before the transmission. We assume the topology illustrated in figure 4. The device addresses for sender, receiver and relay are always given in the format {MAC|IP}_{Send|Recv|Relay}. The receiver listens on port P_Recv, the source port at the sender is unknown. Additionally, we assume that the relay expects incoming data from and send data to the virtual addresses {MAC|IP}_Send_v and {MAC|IP}_Recv_v, respectively. The relay listens on port P_Relay_v_in for the virtual data and forwards to port P_Relay_v_out. Figure 5 illustrates the relay addresses and ports. In the following we also present a sample set for matching and rewriting for each relaying step. Note, for clarification the matching is always based on OSI layer 3 and 4.

1) Original Data Stream: The very first step is to match the original data stream and redirect it to the connected relay. It is checked against the defined flow signature of accepted flows at the relay. If the relaying signature and the packet signature match, the packet header is modified and send out on the interface connected with the relay. The modifications include the source and (virtual) destination information of the OSI layer 2, 3, and 4 as well as the packet's checksum. If they would not be modified, the relays' network stack would discard the incoming packets. Thus, the matching and rewriting rule set is as follows:

	Source Info	Destination Info
Match IP:	IP_Send:*	IP_Recv:P_Recv
Rewrite:	IP_Send_v:*	IP_Relay:P_Relay_v_in

2) Relayed Data Stream: After the original stream has been redirected to the relay, the relay outputs a re-encoded version of the input. This output is transmitted on the same interface as the input comes in, thus it directly hits the OpenFlow node N. This time, the packet's layer 2 and 3 source information correspond to the relays MAC and IP address and the destination information of the packets are purely virtual, that is no device in the net matches this information. Thus, both source and destination header must be modified to ensure a reliable transparently terminated transmission. In fact the rule-set is:

	Source Info	Destination Info
Match:	IP_Relay:*	IP_Recv_v:P_Relay_v_out
Replace:	IP_Send:P_Send	IP_Recv:P_Recv

At this point, the data stream has exactly the same signature as the original data stream and takes its way through the network. Thus, the transparent relaying does not affect any following devices.

3) Original Feedback: In case the receiving application sends feedback to the source, the feedback stream must also be matched and redirected. This is required since the receiver is not directly connected to the sender and thus it potentially uses different connection sequence numbers. This feedback stream flows in the opposite direction. Therefore, the feedback stream comes from the receiver device and hits the OpenFlow node N. Note, that the feedback port (OSI layer 4) assignment must follow a special scheme. Here, we define that the feedback from the receiver is addressed to the senders source IP with port equal to original listen port at the receiving site, but increased by 1. That is P_Sender_FB = P_Recv + 1. The rule-set for the matching and rewriting of the feedback coming from the receiver is as follows

	Source Info	Destination Info
Match:	IP_Recv:*	IP_Send:P_Recv+1
Replace:	IP_Recv_v:*	IP_Relay_v:P_Relay_v_out+1

4) Forwarded Feedback: After the relay has received and processed the incoming feedback packets, they are sent to the attached OpenFlow node N again. Obviously, they are matched and modified again with the following rule-set:

	Source Info	Destination Info
Match:	IP_Relay:*	IP_Send_v:P_Recv+1
Replace:	IP_Recv:P_Recv	IP_Send:P_Recv+1

Note, that the last two matching and modification steps are only required in case the application uses a feedback stream (e.g. TCP).

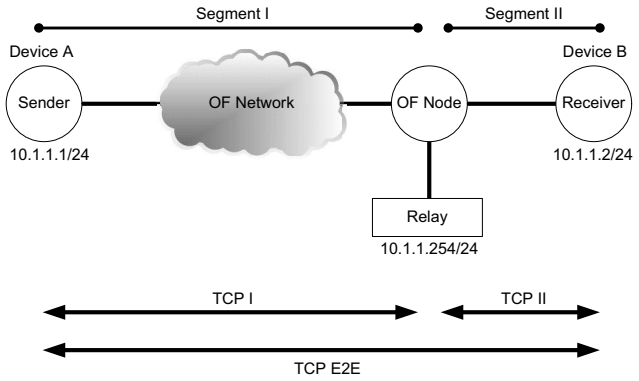

Fig. 6. Evaluation environment.

VI. Evaluation

The evaluation is based on the SDN-based environmental setup illustrated in figure 6. It consists of two client devices (A, B) which are connected via one network path. This network path consists of two network segments I and II, each with individual packet loss rate $p_{I|II}$. Note, that here segment I represents a longer network path including multiple OF nodes. Segment II equals a single network link, e.g. a mobile radio connection. In case a relay is used to intercept the transmission between A and B in this environment it is assumed that it is located exactly between segment I and II. For the end-to-end transmission, the relay is omitted and the network path is considered as continuous. In this scenario, device A is downloading an 850 MB file via TCP (New Reno)/HTTP from device B. The SDN connecting both devices is implemented using OpenFlow 1.0, the network is controlled by the NOX-classic controller. The relay is implemented by the open-source software `socat`[2] with both incoming and outgoing TCP IPv4 socket. Note that `socat` does not incorporate any buffering and thus slows down the incoming pipe in case the outgoing pipe performs at lower speed. During each evaluation process the network conditions are fixed, except for the packet loss probability which is performed by the network emulator `netem`. That is, network segment one is always considered as error-free whereas the second segment has varying loss rates. This equals the aforementioned sample scenario for a typical consumer setup: backbone and mobile radio network. Furthermore, it is assumed that no cross traffic is present on the segments. The evaluation uses a *Wireshark*[3] network capture to count the desired TCP retransmission packets. The quality of the proposed approach is measured as the number of retransmissions used during the sending process. Tables I and II show the results for our benchmarking with varying loss probabilities $p_{II} = \{0.03, 0.05, 0.1\}$ on segment II. As one can see, with the multi-hop approach only segment II is affected by TCP retransmission whereas segment I faces no retransmission packet. With end-to-end both link have to carry retransmissions since the whole connection is treated as one virtual segment. Figure 7 provides an overview of the arithmetic mean of retransmission packets according to the capture. It can be seen that the multi-hop approach clearly outperforms the traditional end-to-end approach.

[2]http://www.dest-unreach.org/socat/
[3]http://www.wireshark.org/

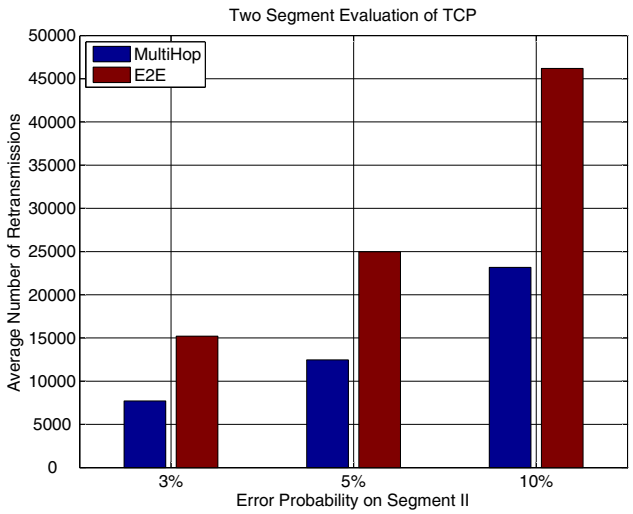

Fig. 7. Evaluation: Average number of retransmission packets per segment.

TABLE I
Evaluation: Average Transmitted Packet Count

	Segment I			Segment II		
Loss p_{II}	3%	5%	10%	3%	5%	10%
MultiHop	0	0	0	15429	24935	46345
E2E	15213	24978	46199	15213	24978	46199

VII. Conclusion

In this paper we propose an analytical view on the difference in data-rate consumption of applications with different transmission schemes. Thereby, we compared the traditional end-to-end transmission scheme and an approach that splits the transmission into multiple smaller segments where each is individually served with error-correction without any timing constraints. It has been shown, that a split transmission always outperforms the traditional approach in terms of lower data rate utilization. Ultimately, we present a relaying scheme for both, uni- and bidirectional transmission, in an SDN environment implemented using the OpenFlow protocol. The present approach can be used for both, multicast and unicast connections for transmissions with no time constraints.

TABLE II
Evaluation of end-to-end and multi-hop Performance.

	End-to-End			MultiHop		
	3%	5%	10%	3%	5%	10%
p_{real}	0.0285	0.044	0.0748	0.0286	0.0438	0.075
RI_{real}	0.0293	0.046	0.081	0.0294	0.0458	0.0811
λ	1.029	1.046	1.081	1.0147	1.0229	1.0405

ACKNOWLEDGMENT

The work presented in this paper was performed in the context of the Software-Cluster project SINNODIUM (www.software-cluster.org). It was partially funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01|C12S01. 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", http://www.opennetworking.org, January 2013

[3] David MacKay, *Information Theory, Inference, and Learning Algorithms*, http://www.inference.phy.cam.ac.uk/itprnn/book.pdf, Cambridge University Press, 2003

[4] Open Networking Foundation, https://www.opennetworking.org, August 2013

[5] Karl, M.; Gorius, M.; Herfet, Th., *Routing: Why less intelligence sometimes is more clever*, IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (ISBMSB), March 2010

[6] 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

[7] Border, J.; Kojo, M.; Griner, J.; Montenegro, G.; Shelby, Z., *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations*, RFC 3135, June 2001