Network Supported Congestion Avoidance in Software-Defined Networks

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

***Abstract*—Large IP networks tend to have a highly heterogeneous structure with many different transmission segments, each endowed with individual characteristics like data rate or delay. Obviously, applications find it difficult to precisely adjust their sending behavior to the underlying topology. Basically, a higher network utilization without congestion is desirable. Server applications demand for reliable information about the available data rate in the network in order to provide a sufficient quality to the clients. Traditionally, this information is completely gathered by source and sink without interaction with the network infrastructure. The rise of software-defined networks (SDNs) enables the integration of the network into this process. In this paper we propose a rate control mechanism for software-defined networks to avoid congestion situations. The approach enables clients to communicate with the network to bidirectionally exchange information about available and occupied data rate. We found that this customized communication approach enables better network load distribution and application transmission reaction - which can be easily implemented with SDNs. Ultimately, the controller finds an efficient forwarding path in the network and the clients have a reliable upper bound for their sending data rate.**

Keywords: Congestion Avoidance, Rate Control, Software-Defined Networking

I. Introduction

According to the Cisco Visual Networking Index (VNI) [1], in 2016 video content with a duration of 1.2 million minutes will cross the global IP networks every second. Many contemporary transmission streams in the Internet are highly inflexible regarding their data rate consumption, which particularly holds for multimedia streams. These streams demand for a reliable lower data rate bound and increase their sending rate up to a specific value. For example consider H.264[2] encoded video streams or the dynamic adaptive streaming via HTTP (DASH)[3] approach. Current networks have extremely complex and imposing structures. Handling and management of these supremely heterogeneous networks are challenging topics since the performance and reliability is subject to multiple factors such as technical characteristics and policy-driven requirements. Traffic Engineering [5] deals with the challenges of evaluating and optimizing the performance of IP networks. Thereby, proactive and responsive approaches are used. The challenge is to cope with different types of traffic. Some have an intrinsic data rate requirement whereas others are completely flexible. Despite this traffic properties all streams must be carried by the network in an effective and efficient way in order to provide reliable and operational communication. The advent of software-defined networks 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-datacenter WAN [4]. The concepts of SDNs are also principally applicable to traditional IP networks: the circuit switching character of matching flows with subsequent forwarding according to predefined rules is already known by Multi-Label Protocol Switching [6]. Obviously, new ideas as information modifications of the lower OSI layers of packets are not available in traditional IP based networks. Despite the many-faceted development, current networking approaches do not provide any mechanism to communicate with applications targeting the goal to precisely adjust the transmission requirements of both, the network and the end-to-end applications. In this paper we propose a communication protocol based on UDP packets which enables the client and network to exchange information about requested, available, and occupied data rate. This information exchange enables all participating parties to communicate information about stream data rates which leads to two different things: On the one hand, the network is able to perform an improved forwarding decision based on the requested and used data rates. On the other hand, the client is not obliged to do congestion control as congestion is avoided beforehand by the announcements of the available data rates through the controller of the SDN. Of course, the client is required to adapt the output data rate to the controller's suggestion. We call this type of rate control *network supported congestion avoidance* (NSCA). This paper is structured as follows: Section II discusses basic concepts of SDNs. Section III introduces the main idea behind rate controlling and section IV presents the NSCA approach. Section V gives an optimzation approach for network stream distribution. Section VI describes an application scenario for the proposed NSCA mechanism and section VII concludes this paper.

II. Software-defined Networking

Software-defined networking describes a recent approach for networking that is also discussed in ITU-T SG 13. It is basically 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 an easy, centralized network configuration. 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. It is also possible to wildcard some flow fields to create more generic matches. When the flow is used to match a stream, 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: 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. Innovations regarding new transport protocols and transportation systems are drastically eased.

III. Rate Control

Currently, there is a set of already deployed rate control mechanisms which mostly are targeted at congestion control and avoidance. Predominant approaches are different forms of the TCP congestion avoidance algorithm, e.g. TCP New Reno [7], TCP CUBIC [8] or Compound TCP [9], and the TCP friendly rate control (TFRC) [10] mechanism. All of them use the perceived losses, round trip time and segment size to determine a sending rate that is fair towards connections that use a compatible scheme. Additionally, there are congestion avoidance mechanisms that include network support and signal the sender of a connection through explicit feedback, e.g. explicit congestion notification (ECN) [11]. Still, most congestion control mechanisms are fed through explicit network characteristics like delay and loss or artificial loss introduced through schemes like random early detection (RED) [12] in case of heavy load. All rate control mechanisms have in common that they need regular feedback from the receiver to calculate the network characteristics and by that their fair share of the available data rate. Media streaming applications tend to require a rather constant data rate to provide an acceptable quality to the end user. More flexible applications are able to send the video in different qualities and apply variable bitrate coding mechanisms as H.264 but their requirement for certain data rate limits remain in between the change of coding parameters. Additionally, frequent changes of coding schemes can disturb the user and impair the experience quality of the application. Therefore, the TFRC algorithm is one of the more suited algorithms for congestion control of media streams as it provides a higher data rate stability. In contrast, the common TCP congestion control algorithms are more unstable in terms of allowed data rate. With the rise of SDNs new possibilities for rate control emerge which are based on the fact that a central controller instance has complete knowledge about the network. With this global view it can assign shares of the available data rate to all flows traversing a specific link. In particular, this is highly interesting for media streaming applications as they need information about the available data rate to prepare a data stream that fits into the given limits. A central controller can provide this information in advance and thus, proactively inform the application to adapt its parameters. We call this type of rate control *network supported congestion avoidance* (NSCA).

IV. NSCA Signaling

Applying NSCA implies that network and client communicate. We propose a communication protocol based on UDP packets which enables the client and network to exchange information about 1) requested, 2) available, and 3) occupied data rate. This communication enables both parties to communicate information about stream data rates and by that allows two things. On the one hand, the network is able to perform an improved forwarding based on the requested and used data rates. On the other hand, the client is not obliged to do congestion control as congestion is avoided through the announcements of the available data rates through the network controller. Of course, the client is required to adapt the output data rate to the network controllers suggestion. Obviously, a traditional method should be provided as a backup procedure.

A. NSCA Signaling Protocol

Figure 1 shows the signaling protocol. The sender communicates with the controller by sending UDP packets to

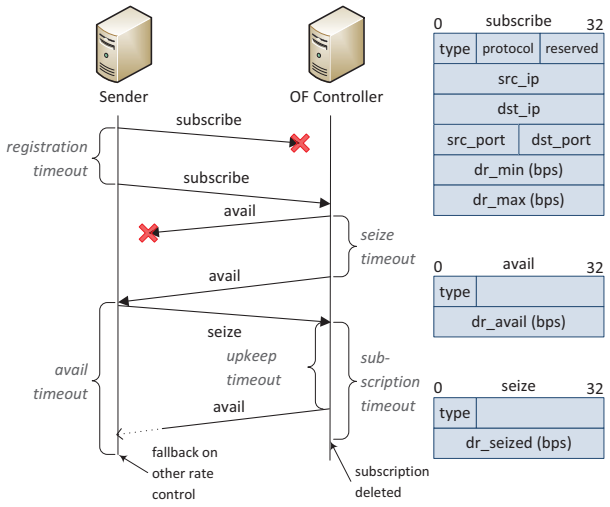

Fig. 1. The signaling between sending client and network controller. The client keeps up a subscription with the controller. The controller announces available data rates and the client responses with the actually seized data rate.

a network specific service address $IP_s : Port_s$ defined by the controller. If present, the controller answers directly to any client requests by responding with a proper UDP reply packet. The sending client must initiate the communication by requesting a subscription for the available data rate from the network. Such a subscription means that the controller sends regular updates about the available data rate to the subscriber. To establish a subscription, the client sends a *subscribe* request which contains information about the minimum and maximum data rates dr_{min}, dr_{max}, and the flow signature of the datastream that is sent. This subscription request is confirmed by the network controller through regular announcements of the data rate that is available for this specific sender dr_{avail}. Finally, the sender must inform the controller about the maximum data rate that will be consumed by the output datastream dr_{seized}. There are several timeouts involved to manage the upkeep of subscriptions and to make the communication reliable. A subscription timeout to_{sub} is managed by the controller for each subscription. All messages that arrive from the subscriber, that is *subscribe* and *seize* messages, refresh the corresponding subscription timeout. Once a subscription timeout elapses, the associated subscription is regarded to be no longer active and is deleted by the controller. To make the exchange of updates reliable the controller uses a seize timeout to_{seize}. Every time an *avail* message is sent to the client the corresponding seize timeout is started. Once this timeout is triggered, the *avail* message is sent again. Also once the *seize* message is received an upkeep timeout to_{upkeep} is started after which the *avail-seize* cycle is repeated. Similarly to the controller, the sending client needs to manage some timeouts too. The first one is the registration timeout to_{reg} which is started after a *subscribe* message is sent to the controller. Once the registration timeout expires, the *subscribe* message is resent. After a subscription is established the sender starts the avail timeout to_{avail} after each received *avail* message. If no *avail* message is received until the timeout expires, the controller is assumed to be unresponsive and a fallback to traditional congestion control mechanisms must be performed. Clearly, the client should continuously try to reach the controller in case it becomes responsive again.

B. Client Workflow

Figure 3 shows a flow chart with the behavior of the sending client. It is divided into two main parts: the data sending loop on the left and the event handling loop on the right. After start-up a *subscribe* is sent to initiate the communication with the network controller. This is followed by setting $nsca$ to $false$ and entering the data loop to send data with a traditional rate control mechanism to avoid unnecessary delay at transmission start up. Basically, the data loop describes that the client sends data with the current rate control, either network controlled or an end-to-end rate control, until the transmission is stopped or an event has been triggered. On the other hand, the event loop describes the actions in case of arrival of an *avail* packet, or a timeout for either an *avail* packet from registration or the regular updates by the controller. If an *avail* packet is received, dr_{seized} is determined, then the NSCA is activated by setting $nsca$ to $true$ and the data loop is reentered if no further event must be processed. Note that all *avail* packets by the controller will reset dr_{seized}. If one of the timeouts elapses, a new subscription is sent and the client continues to use a different rate control mechanism to send data. Notice, that dr_{seized} is set to zero if $dr_{avail} < dr_{min}$. This describes the case that not enough resources are available to transmit a data stream whose flexibility is limited. Once the transmission is over, the subscription will automatically time out at the controller.

C. Controller Workflow

A similar flow chart for the controller is presented in figure 2. The controller holds a cache of currently active subscriptions for which it needs to create flows through the network and determine the maximum possible data rates. The *setup* is done on a regular basis and triggered by an event which is scheduled after the controller starts. In the beginning, the only event that can happen is either the setup event or the arrival of a packet. When a *subscribe* packet arrives, a new subscription is created or an existing subscription is updated and the timeout for this subscription is set. In case of a *seize* packet in addition the timeout to_{seize} is deleted if the subscription is still valid. The *setup* workflow is executed on a regular basis for all setup events that are triggered. It consists of determining the available data rate for each flow, allocating the flows through the network, and sending the *avail* packets to the clients which have a valid subscription. A detailed description of the *setup* procedure can be found in section V. When and how often the *setup* occurs can vary depending on e.g. the performance of the available controller hardware. The remaining events are the timeouts for either subscriptions or expected *seize* packets which lead to the deletion of the

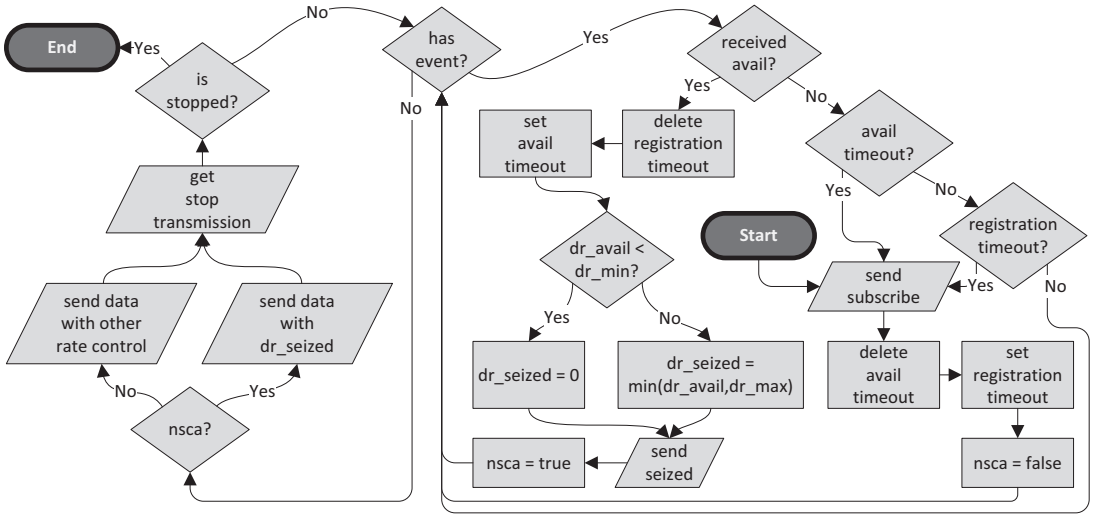

Fig. 3. A flow chart representing the behavior of the sending client. In case no compatible network controller is available the sender will fall back on traditional rate control.

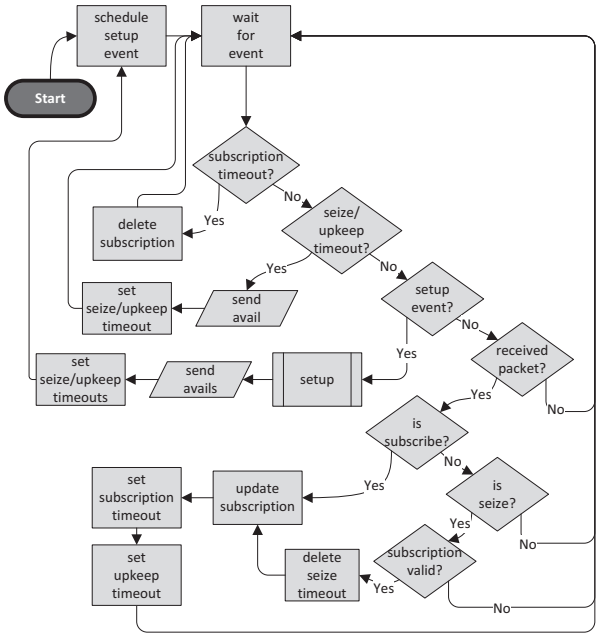

Fig. 2. A flow chart representing the behavior of the network controller.

corresponding subscription or a repetition of the corresponding *avail* packet, respectively.

V. Network Rate Control Service

The Network Rate Control Service (NRCS) calculates the network-wide optimum stream distribution. Here, we assume that all streams in the network are NRCS based, that is, they always limit their rate according to the network recommendation. Another assumption is that the maximum link capacities of all links in the network are known to the controller. The NRCS described here is part of the controller workflow illustrated in Figure 2 and denoted as *setup*. First of all, we introduce some basic definitions that ease the handling of the problem. Afterwards we present the optimization problem and formulate a possible solution. For this scenario, we assume the presence of n streams where all streams s_i are in the set $S = \{s_1, \ldots, s_n\}$. Each stream $s \in S$ has already announced (via the *subscribe* message) its minimum and maximum data rate requirements which can be represented as the interval $I(s_i) = [dr_{min}(s_i), dr_{max}(s_i)]$. The following actions are performed for all streams $s_i \in S$: For a better optimization solution, we introduce an interval tuning factor k that splits $I(s_i)$ into multiple equally sized portions with the step-size:

$$\Delta I(s_i) = \frac{dr_{max}(s_i) - dr_{min}(s_i)}{k}$$

Thus, we obtain a set of data rates for stream s_i:

$$\begin{aligned} \mathrm{DR}_k(s_i) = &\{dr_{min}(s_i), dr_{min}(s_i) + \Delta I(s_i), \\ &\ldots, dr_{min}(s_i) + k \cdot \Delta I(s_i)\} \\ = &\{dr_0(s_i), dr_1(s_i), \ldots, dr_k(s_i)\} \end{aligned}$$

Note that k influences the performance, granularity and complexity of the calculations. Obviously, larger values for k lead to a higher number of possible data rates whereas a smaller value of k leads to a coarser optimization result. k can also depend on additional factors as the transport protocol (UDP, TCP). Then, for each $dr_j(s_i) \in \mathrm{DR}_k(s_i)$ the set of network paths between source and target is determined. But before applying a modified (no visited flag and no exit if node was found the first time) breadth-first search (BFS) that finds all paths between the source a and the desired target node b the network is pruned. That is, all links l are ignored in the BFS that satisfy the condition $cap(l) < dr_j(s_i)$. Therefore, only links with valid capacity values are included in the BFS result and the computational complexity decreases. The runtime complexity of a BFS is generally $O(|Nodes| + |Links|)$. Besides this, the computational complexity of our approach

increases with the number of streams in the network. In this paper we only focus on basic, non-optimized solution that leaves room for further improvements. Afterwards, the BFS provides a set of paths for stream s_i for each data rate $dr_j(s_i)$: $P(s_i, dr_j(s_i)) = \{p_1(s_i), p_2(s_i), \ldots\}$. The set of paths varies for each data rate because paths with a lower bottleneck capacity are only available for the stream if the data rate is small enough. The last step is to build the stream-setup tuple set $\Gamma(s_i)$ for the stream s_i with $z = |\Gamma(s_i)|$ being the cardinality of Γ:

$$\begin{aligned}\Gamma(s_i) &= \{\gamma_1(s_i), \gamma_2(s_i), \ldots, \gamma_z(s_i)\} \\ &= \bigcup_{\substack{1 \le i \le n \\ 0 \le j \le k}} [P(s_i, dr_j(s_i)) \times \{dr_j(s_i)\}] \\ &= \{(p_j, d_j) | 1 \le j \le z\}\end{aligned}$$

Now, assume for all $s_i \in S$ the stream-setup tuple set $\Gamma(s_i)$ is given. We define a network-setup tuple $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$, where each entry π_i corresponds to an entry $\gamma_i \in \Gamma(s_i)$. The main challenge now is to find an allocation for Π that optimizes the network resource utilization. More precisely, with $\gamma_i = (p_i, d_i)$, and l as a link in the network, the problem is described as follows. Find all allocations Π that build the following set NS of network setups:

$$NS = \{\Pi \mid \forall l \in N : cap(l) \ge \sum_{i=1}^{n} d_i \cdot x_{i,l}^*\}$$

where the shared link indicator variable $x_{i,l}^*$ is defined as

$$x_{i,l}^* = \begin{cases} 1, & l \in p_i \\ 0, & \text{otherwise} \end{cases}$$

From this set NS take the Π that maximizes a predefined gain function $g(\Pi)$. This gain function can have different forms but for the sake of simplicity it can be defined as:

$$g(\Pi) = \sum_{i=1}^{n} d_i$$

This means we define the optimization of the network resource utilization as the process of providing as much data rate to the clients as possible. Algorithm 1 shows a full-search approach to find the optimum distribution. d^* defines the currently highest data rate found by the algorithm and Π^* reflects the corresponding network-setup. Lines 4 to 7 build the main loop to check all possible combinations. Note that the descending search order leads to results earlier, since all the stream-setup tuple sets have the highest data rates at their end. In line 10 the algorithm checks if the paths of the streams have links in common. If so, it is checked if the available link capacity is sufficient to handle all stream data rate demands. In case the capacity is not high enough, the current combination is omitted and the next combination is considered (remember the shared link indicator $x_{i,l}^*$ from above). In line 22 it is checked if the sum of provided data rates exceeds the currently highest one (This check can be replaced through the application of a different gain function). If so, the value and the network-setup are stored in d^* and Π^*, respectively.

Algorithm 1 Network-Optimization

```
1:  procedure SOLVE(Γ(s_1), Γ(s_2), ..., Γ(s_n))
2:      d* = 0
3:      Π* = (∅, ∅, ..., ∅)
4:      for each γ_1 ∈ Γ(s_1) in desc. order do
5:          for each γ_2 ∈ Γ(s_2) in desc. order do
6:              ...
7:              for each γ_n ∈ Γ(s_n) in desc. order do
8:                  let (p_i, d_i) = γ_i
9:                  let L = ⋂_{k=1}^{n} p_k
10:                 if L ≠ ∅ then                    ▷ shared links
11:                     congested = false
12:                     for each l ∈ L do
13:                         if cap(l) < Σ_{k=1}^{n} d_k · x*_{k,l} then
14:                             congested = true
15:                             break
16:                         end if
17:                     end for
18:                     if congested == true then
19:                         continue
20:                     end if
21:                 end if
22:                 if Σ_{k=1}^{n} d_k > d* then
23:                     d* = Σ_{k=1}^{n} d_k
24:                     Π* = (γ_1, γ_2, ..., γ_n)
25:                 end if
26:             end for
27:         end for
28:     end for
29: end procedure
```

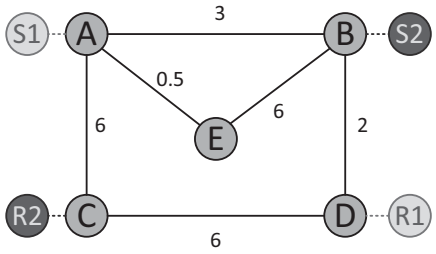

Fig. 4. The setup for network in of our example. The numbers on the links state their capacity.

VI. Application Scenario

In this section we present an example in which the NRCS is applied. Two senders S1 and S2 want to send their streams s_1 and s_2 through the network to the receivers R1 and R2, respectively. The corresponding network including the link capacities is shown in figure 4. We assume bidirectional links in the network where both directions provide the same capacity. The senders subscribe to the NRCS by stating their minimum and maximum possible data rates given in table I. Given this information the NRCS starts to calculate a network setup that satisfies the data rate requirements of the streams with the available network resources. First, the NRCS

TABLE I
MINIMUM AND MAXIMUM DATA RATES OF THE STREAMS IN THE EXAMPLE.

stream	dr_{min}	dr_{max}
s_1	2	4
s_2	1	3

TABLE II
A SELECTION OF NETWORK-SETUPS FOR THE EXAMPLE SCENARIO. BOLD AND GRAY ROWS ARE POSSIBLE COMBINATIONS.

index	$dr(s_1)$	$p_i(s_1)$	$dr(s_2)$	$p_i(s_2)$	$dr(s_1)+dr(s_2)$
..	..	..	..	..	..
9.	**2**	$\boldsymbol{p_1}$	**2**	$\boldsymbol{p_1}$	**4**
10.		p_1		p_2	
11.		$\boldsymbol{p_2}$		$\boldsymbol{p_1}$	
12.		$\boldsymbol{p_2}$		$\boldsymbol{p_2}$	
13.	**3**	$\boldsymbol{p_2}$	**2**	$\boldsymbol{p_1}$	**5**
14.		$\boldsymbol{p_2}$		$\boldsymbol{p_2}$	
15.	**4**	$\boldsymbol{p_2}$	**2**	$\boldsymbol{p_1}$	**6**
16.		$\boldsymbol{p_2}$		$\boldsymbol{p_2}$	
17.	**2**	$\boldsymbol{p_1}$	**3**	$\boldsymbol{p_1}$	**5**
18.		$\boldsymbol{p_2}$		$\boldsymbol{p_1}$	
19.	**3**	$\boldsymbol{p_2}$	**3**	$\boldsymbol{p_1}$	**6**
20.	4	p_2	3	p_1	7

calculates all possible paths over which the streams can be routed through the network. For each stream there are two possible paths assuming the minimum requested data rate. These paths are $p_1(s_1) = (AB, BD)$, $p_2(s_1) = (AC, CD)$, $p_1(s_2) = (BA, AC)$, $p_2(s_2) = (BD, DC)$. As each of these paths has an associated maximum data rate, only a reduced set of stream-setups makes sense for each stream. For example, routing over $p_1(s_1)$ is meaningful only if the data rate of stream s_1 is smaller or equal 2. Combining all sensible stream-setups of s_1 and s_2 results in 20 network-setups from which a selection is shown in table II. Note that the number of network-setups is strictly dependent on the data rate granularity, the number of possible paths according to these data rates, and the overall number of streams in the network. To keep the complexity low, it is possible to separate the network into smaller, individually controlled network parts. Obviously, a higher degree of separation decreases the optimum routing result but decreases the computational complexity at the same time. In this table you can see that the network-setup with the maximum data rate of both streams is not feasible as the shared link AC provides a capacity of only 6 whereas the combination of both streams would require a capacity of 7. Nevertheless, there are several possible combinations from which the NRCS can choose according to an arbitrary policy. In our case, the goal is to maximize the network utilization so the network-setups with the largest overall data rate $dr(s_1)+dr(s_2) = 6$ are selected. In case a secondary goal is to provide fairness among the streams, the NRCS can choose a network-setup where the variance of the stream data rates is minimal. In this example this would result in the selection of network-setup number 19. Choosing a network-setup can even be based on contract agreements as streams can be mapped to senders and/or receivers by means of their flow signature.

VII. CONCLUSION

In this paper we proposed NSCA which represents an innovative way of rate control for congestion avoidance in software-defined networks. We developed a signaling protocol that is applied by the controlling plane of the SDN to provide a network rate control service (NRCS). The NRCS can be utilized by sender devices to obtain the available data rate. We presented an algorithm to find a distribution of allowed data rates and forwarding paths in a multi stream scenario which can be tuned to accommodate various goals, such as maximum network utilization, fairness, or QoS. Additionally, this proactive scheme avoids congestion entirely before it happens. Also, a traditional scheme can be used for rate control as fallback in case the NRCS is unavailable. Further improvements can be achieved with more sophisticated optimization algorithms and can be part of future research in that area.

ACKNOWLEDGEMENT

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] 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.
[3] ISO/IEC 23009-1:2012(E): Information technology - Dynamic adaptive streaming over HTTP (DASH), International Organization for Standardization, Geneva, Switzerland.
[4] Google, "Inter-Datacenter WAN with centralized TE using SDN and OpenFlow", January 2013
[5] Awduche, D., Chiu, A., Elwalid, A., Widjaja, I., and X. Xiao, "Overview and Principles of Internet Traffic Engineering", RFC 3272, May 2002.
[6] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, January 2001.
[7] Floyd, S., Henderson, T., and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782, April 2004.
[8] Sangtae Ha, Injong Rhee and Lisong Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", ACM SIGOPS Operating System Review, Volume 42, Issue 5, July 2008, Page(s):64-74, 2008.
[9] Tan Kun, Jingmin Song, Qian Zhang, Sridharan, M., "A Compound TCP Approach for High-Speed and Long Distance Networks", INFOCOM 2006, 25th IEEE International Conference on Computer Communications Proceedings, April 2006.
[10] Handley, M., Floyd, S., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 3448, January 2003.
[11] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
[12] Floyd, S., Jacobson, V., "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, August 1993.
[13] Bellovin, S., "The Security Flag in the IPv4 Header", RFC 3514, April 1 2003.