SDN-based Application-Aware Networking on the Example of YouTube Video Streaming

Michael Jarschel, Florian Wamser, Thomas Höhn, Thomas Zinner, Phuoc Tran-Gia
University of Würzburg, Institute of Computer Science, Würzburg, Germany.
Email: {michael.jarschel,florian.wamser,thomas.hoehn,zinner,trangia}@informatik.uni-wuerzburg.de

***Abstract*—Application-Aware Networking is a promising approach to provide good application quality to users in scenarios with limited network resources, like today's access networks. With SDN, a particularly interesting method to enable flow-based traffic management in networks has become available. In this work we take a look at how a specific application, i.e., YouTube Streaming, can benefit from such an SDN-based Application-Aware Network. We implement and investigate an approach based on Deep Packet Inspection (DPI) and one based on direct information input from the application in an OpenFlow testbed in order to show, how these different types of application information can be exploited to enhance the Quality of Experience (QoE). Furthermore, we determine the overhead caused by each of the presented approaches.**

***Keywords*-Application-Aware Networking, SDN, Software Defined Networking, OpenFlow, DPI, Deep Packet Inspection**

I. Introduction

The requirements for network applications are diverse and today's networks try to support them based on Quality of Service (QoS) parameters. However, the performance of a specific application cannot be determined by simply relying on QoS metrics [1]. Instead, a good application quality, e.g., the video quality or short waiting times, is the metric by which a user quantifies his/her Quality of Experience (QoE). Therefore, a major challenge for future networks is to dynamically adapt to QoE demands of the applications in the network. This is especially true for networks with limited resources, like today's access networks. Application-Aware Networking is a way to provide a good application quality to users of these networks.

The introduction of Software-Defined Networking (SDN) opens a path towards the realization of this approach. By introducing an external and programmable network control plane, SDN creates a flexible, adaptable, and open interface to the network, the "Northbound-API". It enables the exchange of application information with the network. This in turn can be leveraged to augment the network management to improve the user QoE. The challenge here is to determine which kind of information should be how often exchanged.

In this paper we examine how different kinds of information, such as per-flow parameters, application signatures, or application quality parameters can support a more effective network management in an SDN-enabled network. Application information and related QoS levels offer greater flexibility in terms of supporting QoE than hard QoS parameters. However, using them may require an overhead of signaling effort compared to management at the network level. Therefore, we take a look at the trade-off between the QoE improvement due to more detailed application information and corresponding signaling overhead. All approaches are emulated in an SDN-enabled testbed for the application of YouTube streaming. We use the YouTube quality monitoring tool YoMo [2], which monitors the buffer filling level and the occurrence of playback stalling to quantify the impact each approach has on the YouTube QoE.

The remainder of this paper is structured as follows. In Section II we give background information to SDN and Application-Aware Networking and discuss related work. We then introduce our testbed setup and scenarios in Section III. Section IV discusses the different approaches with the corresponding experimental results. Finally, we derive our conclusions in Section V.

II. Background and Related Work

Currently, the prevalent idea in networking for improving the quality of a service for the end-user is to differentiate traffic flows into Quality of Service levels. For this purpose, different QoS classes are defined according to the expected type of traffic in the network and applications with similar needs are assigned to them. These classes ensure a minimum reserved traffic rate according to the QoS parameters of the application type.

However, a QoS-based provisioning alone is often not sufficient to provide an acceptable application quality. This is especially the case for applications with time-dynamic QoS requirements. For example, due to video encoding, download patterns, or user behavior an application may not have a fixed demand for bandwidth. Instead, bandwidth is required depending on the application state. SDN provides an interface to convey this application state to the network. This allows the network control plane to optimize the flow of traffic according to the information available.

A. Application-Aware SDN

In an SDN-enabled world as we see it, new open interfaces exist between the application, the data-plane, and the

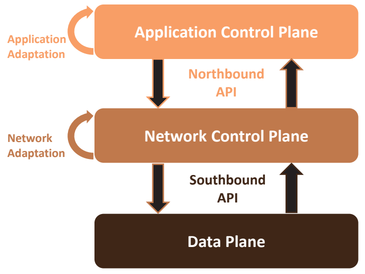

Figure 1. Application-Aware SDN Architecture

control-plane. These are illustrated in Figure 1.

The interface between data- and control-plane is called the ”Southbound-API”. It enables the externalization of the control plane from the forwarding device to a logically-centralized network control plane, often simply called ”controller”. As a software entity, the controller can be freely programmed and adapted to the network according to the operator’s requirements. Currently, the most popular realization of this interface is OpenFlow [3], which we use for our experiments. While the Southbound-API is an important component of SDN, from our point of view, the significant additional value of SDN lies within the ”Northbound-API” interface between the network control plane and what we call ”application control plane”, i.e. applications running on top of or interacting with the network itself. This enables the exchange of information about the application and network state, respectively. Curtis et al. [4] suggest an optimized data center flow scheduling by notifying an OpenFlow-like controller about elephant flows detected at the hosts’ socket buffers. In [5] Das et al. demonstrate how SDN-based aggregate routing can be adapted with the QoS parameters of applications in mind. Jarschel et al. [6] show how a pre-notification of the network control plane in case of a virtual machine migration can serve to maintain service. We are going one step further by also taking the actual application quality and state over time into account to maintain a good service quality for the customer.

B. *Technical Details on YouTube Streaming*

YouTube, one of the most important VoD platforms, provides mainly small to medium sized video clips in different qualities. The default video compression format is H.264/MPEG-4 Advanced Video Coding (AVC). In order to watch a video, the user opens the YouTube web page where an HTML-5 or Adobe Flash player is embedded for video playback. The video player requests the video data from a YouTube streaming server in the Internet using the HTTP protocol. YouTube uses progressive video streaming which means that the video is already played out, while the client downloads the content into a buffer or a temporary file in the background. If the buffer is sufficiently filled, a smooth video playback can be guaranteed. If the buffer is empty, the video playback is interrupted and stalling occurs. According to [7], [8], stalling is the dominating factor of the QoE for online video streaming, clearly exceeding the significance of video resolution. Hence, a simple mapping of a QoS parameter such as throughput to YouTube QoE is difficult, as the QoE depends on the buffer level and video encoding. This complexity makes YouTube streaming a good candidate for the Application-Aware SDN approach.

III. Scenario and Testbed Setup

Our Application-Aware SDN testbed emulates a path selection scenario for an access network provider. The access network provider, e.g. a mobile network operator, transmits the data of its customers over multiple leased lines to the Internet. The goal of the provider is to use these lines as efficiently as possible, i.e., as few lines as possible should be rented as long as the QoE of the user does not suffer.

The provider has chosen an OpenFlow-enabled device as termination point for several access connections to its customers. While the provider does have exclusive last mile-access to the customers, the upstream connectivity belongs to a different ISP. Therefore, the OpenFlow device is connected via leased virtual channels across the WAN to a second OpenFlow-enabled device in the provider’s Internet backbone. A customer of the provider is watching a YouTube video, while other customers run file downloads or surf the web. The provider is interested in providing a good quality of experience to the YouTube user, while at the same time not overextending its leased resources.

A. *Testbed Setup*

Figure 2 shows our testbed setup for the reference case. As OpenFlow-enabled devices at the access and provider edge, we use two Pronto 3290 switches [9] running PicOS 1.6.1. Both are configured for out-band management and are connected via their management interfaces to an HP ProCurve 1810-24 switch, forming the management network. A Dell PowerEdge 860 server is used as controller host and is also connected to the HP switch. As controller software, we are using the Floodlight controller [10] from BigSwitch running our own modules. The ”virtual” provider connections are represented by five links between the two switches using Cat-5 cabling. The physical ports on the switches for these five links are set to 10 Mbps link speed. The ”provider switch” is connected to a Cisco router, which serves as Internet gateway for our testbed. The YouTube user is represented by a standard PC running Ubuntu Linux. The browser used to access YouTube is Mozilla Firefox running our YoMo plugin. When the browser is directed to play a YouTube video, YoMo, among other things, is able to identify the TCP-flows used for the transmission as well as track the buffered and current playtime in the YouTube player. Since the QoE of YouTube depends on stalling [7], [8], monitoring the buffered playtime gives us an indication

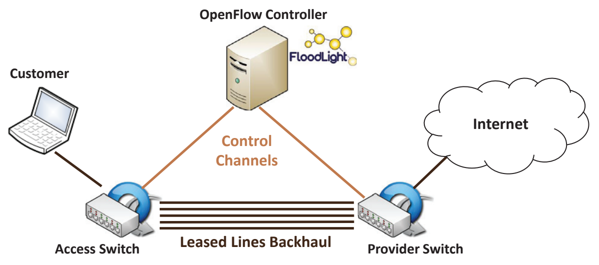

Figure 2. Testbed Setup

whether the current performance offered by the network leads to a QoE degradation for the user or not.

B. Experiments

In the following, we discuss the differences between each of the experiments as well as their purpose. The start of the YouTube video coincides with the start of each experiment. The video is played out with a resolution of 480p by default.

Reference Experiment: In this experiment only YouTube traffic to a single client is transmitted. The controller chooses one of the five available links at random to transfer the flow. The YouTube traffic can use the full 10 Mbps available on that link. This experiment gives us a baseline in terms of the available buffered playtime we can expect under optimal conditions.

Reference Experiment with Interfering Traffic: For this experiment two additional PCs are connected to the testbed. One is connected to the access switch, the other to the provider switch. We use Iperf [11] to generate traffic between those two machines in order to emulate other users on the network. In addition to the YouTube traffic, 20 TCP flows are started sequentially after 60 seconds with a flow inter-arrival time of one second. The controller directs all traffic via only one of the five links, which gives us a worst case approximation.

Round-Robin Path Selection: The testbed setup for this experiment remains the same as in the previous case. Once again 20 TCP flows are generated. However, this time the controller can use more than one link. It does so by directing each new flow to a different link in a round-robin fashion. This experiment represents our naive load-balancing approach.

Bandwidth-Based Path Selection: The same traffic and testbed as in the previous experiments is also used here. The controller is still able to use all links. Links are selected by their currently used bandwidth. When a new flow arrives, the controller determines the least loaded link and directs the flow to it. At the same time the controller checks the bandwidth required by each of the flows every second via the switches' flow table counters. If there is a link with free capacity available, the controller will then redirect the largest flow in terms of bandwidth consumption from a loaded link to the free link. In order to avoid constant redirection, this can only happen once every ten seconds for a specific flow.

Deep Packet Inspection: We extend the testbed by a machine performing Deep Packet Inspection (DPI). The machine is connected to the management network and can be contacted by the controller. The experiment parameters are the same otherwise. In this experiment, the controller directs all traffic via one link. The first ten packets of each flow, are mirrored to the controller, which then sends them to the DPI machine running a combination of TShark, the console version of Wireshark [12], and several filter rules based on regular expressions. The DPI informs the controller about the nature of the flow. If a particular flow is a YouTube video, the controller will redirect the flow to another less congested link.

Application-Aware Path Selection: Finally, in this experiment, we leverage the information YoMo provides us with as input for the controller. The experiment is identical to the previous one, except that 50 TCP flows are generated to create a high load scenario and the machine used for DPI in the previous experiment is now used to receive application information containing the current YouTube buffer level and flow information. We call this machine the ”application station”. When the buffer level gets below a certain threshold, the application station informs the controller that an action is required for a particular flow in order to maintain the QoE for the user.

IV. Measurement Results

In this section, we discuss the measurement results of our experimental investigation. All experiments were repeated five times. However, for the sake of visualization, only one representative run is depicted. Each experiment has a duration of 420 seconds. The used video[1] has a mean data rate of 2.6 Mbps with standard deviation of 250 kbps.

A. Reference Experiment

The upper curve in Figure 3 shows the pre-buffered playtime of the YouTube player at the client in seconds over the duration without interfering traffic. It can be seen that a pre-buffered playtime of about 55 seconds is reached within the first 10 seconds. Thus, 55 seconds playback can be achieved without further data. While the video is played out, the buffer decreases but is constantly refilled so that the buffer maintains a stable level. This is as expected for the reference experiment. The buffer level never drops significantly and, most importantly, it never reaches zero, which would cause the video to stall.

B. Reference Experiment with Interfering Traffic

The repetition of the reference experiment with interfering traffic yields a different result as is illustrated in Figure 3,

[1]YouTube Video: ”Waterfall” 90mins ”Sleep Video” Bull Creek; http://www.youtube.com/watch?v=WZtn2n51Xrw

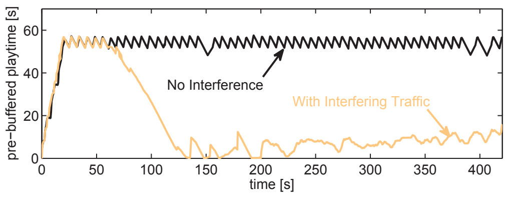

Figure 3. Buffered Playtime (1 Link), No Interference and 20 Additional TCP Flows

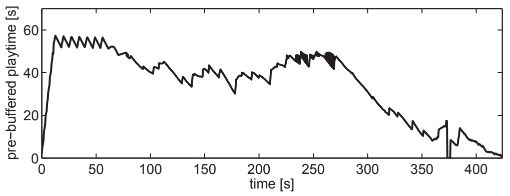

Figure 4. Buffered Playtime (5 Links, 20 Additional TCP Flows, Round-Robin Path-Selection)

which again shows the pre-buffered playtime in seconds over time. Up to the point when the interfering traffic starts, the behavior is exactly the same as in the reference experiment. However, with the reduced bandwidth available, the buffer level continuously falls as the video is played out until it is emtpy and the video stalls at about the 140s mark. At about 200s the YouTube player automatically reduces the default resolution of 480p to 360p, and therewith the video bit rate. This enables the video to be played out again with less stalling albeit in a lower quality. The buffer, however, does not recover and stays at a low level of about 10s pre-buffered playtime.

C. Round-Robin Path Selection

The approach of balancing the flows across multiple links in a round robin fashion should naively improve the situation for the YouTube user compared to the one link scenario. However, this is not necessarily the case with heterogeneous traffic as is shown in Figure 4. After the initial undisturbed phase, the YouTube buffer once again can not maintain its high level. However, this time it is not immediately emptied and the video keeps playing. As the controller assigns all flows in a round robin-fashion but can not tell, which flow is an ”elephant” and which is a mouse, the bandwidth distribution can be very uneven. This eventually also comes to haunt our YouTube video at about 270s after the start of experiment when it has to share its link with multiple high bandwidth flows. Subsequently, the buffer is drained and the video stalls yet again.

D. Bandwidth-Based Path Selection

Taking the used bandwidth per flow into account is the next logical step from our round-robin approach that suffered from uneven bandwidth distribution. However, it fares little

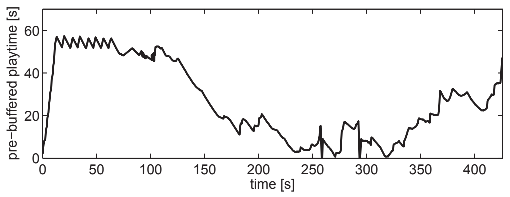

Figure 5. Buffered Playtime (5 Links, 20 Additional TCP Flows, Bandwidth-Based Path-Selection)

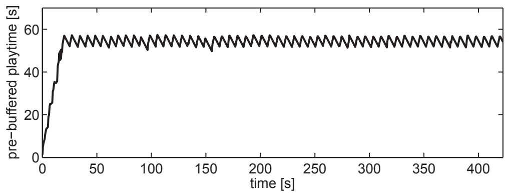

Figure 6. Buffered Playtime (5 Links, 20 Additional TCP Flows, DPI-Based Path-Selection)

better in terms of YouTube streaming performance as can be seen in Figure 5. This is due to fact that all flows, except the YouTube flow, are ”elephants” and try to use the maximum bandwidth available. When the interfering traffic starts at 60 seconds, the buffer begins to decrease. While this is not as swift as in the one-link scenario, it steadily decreases and eventually reaches a stalling event. At this point YouTube switches again to a lower resolution and the video is able to recover.

E. Deep Packet Inspection

The previous experiments show, that network information alone is not sufficient to provide the YouTube user with a good performance. However, using deep packet inspection, we can identify the YouTube traffic in the network and prioritize it. This approach yields the desired success as is shown in Figure 6. The YouTube pre-buffered playtime behaves the same way as in our reference scenario without any interfering traffic. However, the overall usage of the available network resources is reduced as depicted in Figure 7. Here, the used bandwidth in Mbps over time is shown. Initially, there is a spike up to the maximum utilization of 10 Mbps on Link1. This is due to the YouTube buffer ramping up. After about 20 seconds the deep packet inspection has identified the traffic as YouTube video and notified the controller, which redirects the video to its own dedicated Link2. When the interfering traffic starts at 60s, it remains on Link1, not able to influence the YouTube stream. Since the DPI can not provide application state information all links are reserved for YouTube streams, which in this case results in a network resource utilization of just about 15 percent on average.

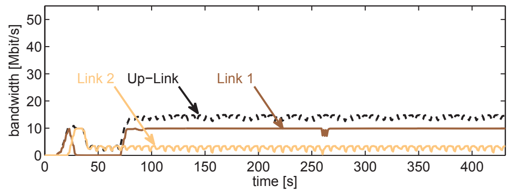

Figure 7. Used Bandwidth (5 Links, 20 Additional TCP Flows, DPI-Based Path-Selection)

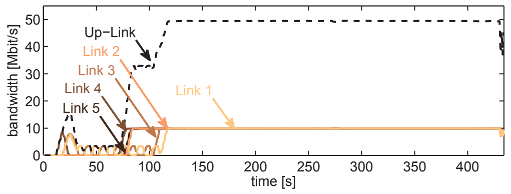

Figure 9. Used Bandwidth (5 Links, 50 Additional TCP Flows, Application-Aware Path-Selection)

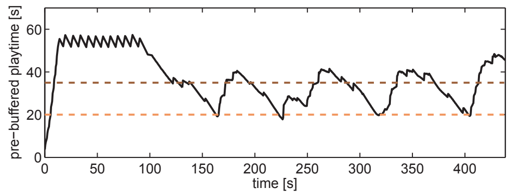

Figure 8. Buffered Playtime (5 Links, 50 Additional TCP Flows, Applicaton-Aware Path-Selection)

F. Application-Aware Path Selection

While the deep packet inspection approach has already enabled us to provide a good experience to the user, we also wasted a lot of bandwidth as the dedicated link for the YouTube video is only slightly used after the initial ramp-up of the buffer and the other dedicated links remain empty. It would be beneficial for the network only to use the extra resources when there actually is a problem and return to normal operation when it no longer persists. This is where the benefits of the SDN northbound interface come into play. By leveraging this interface, we can implement application-state awareness in the network. The benefits can be seen in Figure 8. With all traffic on one link, the buffer level of the YouTube video starts to decrease, like we have seen in IV-B. When it reaches a threshold of 20s pre-buffered playtime, the controller is triggered and it redirects the YouTube traffic to a less-loaded link. However, this time this is not a dedicated link. It can be used by other traffic. Therefore, once the YouTube video has reached a buffered playtime of 35 seconds, the controller can use more capacity on the link for other traffic until the video again reaches its lower playtime threshold. As we can see in Figure 9, all links in this experiment are fully loaded. Despite of this, the YouTube user still experiences a good quality using the Application-Awareness approach.

G. Resource-Overhead

All in all, two OpenFlow switches are required for these approaches in addition to the controller. For deep packet inspection and application-aware networking another machine is required to gather information about the running applications. Furthermore, all of the described methods to improve the performance for the YouTube user cause a certain overhead on the control plane. In the following, we describe said overhead for each approach and determine its efficiency in terms of resource consumption. As a gauge for the efficiency ρ of resource utilization, we use the ratio of bandwidth used on average once the interfering traffic has started,

$$\rho = \frac{Mean(UsedBandwidth)}{AvailableBandwidth}.$$

Round-Robin Path Selection: The simplest solution with round-robin flow scheduling also has the least overhead. Here, only the OpenFlow Packet-In, Packet-Out and Flow Mod messages have to be transmitted via the control channel. No additional traffic and components are necessary. For this approach all resources are used, therefore $\rho \approx 1$.

Bandwidth-Based Path Selection: For the bandwidth-based approach, more overhead in terms of control channel traffic is required compared to the round-robin approach. The controller needs to periodically query the flow table counters in the switches to determine the current bandwidth utilization of each flow and link. Additionally, the balancing of bandwidth usage causes more flow redirection operations, which increases the number of sent Flow Mod packets and the CPU load on the switches. Again all available resources are used with $\rho \approx 1$.

Deep Packet Inspection: The deep packet inspection approach requires an additional, potentially heavy loaded, computing resource in the control plane to perform the packet analysis. Furthermore, the first ten packets of each flow have to be mirrored. We use the control channel for this. The bandwidth utilization on the control channel in the DPI case shows an increase of bandwidth utilization to just under 1 Mbps when packet mirroring occurs at the beginning of the experiment. This is double the required bandwidth used by normal reactive flow setups, which peak at about 0.5 Mbps. However, compared to the bandwidth-based approach, no real-time querying of the switches is necessary. A general problem of DPI is also that the application signatures have to be constantly updated, which requires additional expenses. As there is no access to the application state in this approach, all links except one have to be dedicated lines for potential YouTube flows. As in our scenario only one YouTube flow is actually transmitted a lot of bandwidth is left unused leading to a ρ of just 0.15. Using only one line for YouTube flows

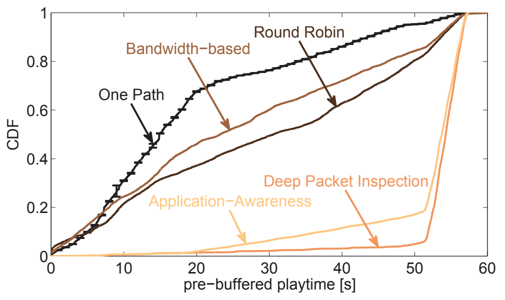

Figure 10. Cumulative Distribution Function for the Pre-Buffered Playtime (20 Additional TCP Flows)

would increase the bandwidth usage to $\rho \approx 0.85$ which is still less compared to the other scenarios.

Application-Aware Path Selection: Like DPI, the application-state-aware approach also requires an additional computing instance to receive and filter the application information for the controller. While this method puts no significant overhead on the control channel, it requires the exchange of information of the application station instance and the served clients. This may cause a significant amount of traffic, if the application state changes constantly. Furthermore, an additional software component is required at the client to monitor the application. This can be a part of the application itself or, as is the case with us, be a plugin for the software that should be monitored. With the benefit of application state information, all resources can once again be used leading to a $\rho \approx 1$.

H. *Quantifying the Results*

Figure 10 shows the cumulative distribution functions of the pre-buffered playtime for five experiment runs of our approaches once interfering traffic has started. For the sake of readability the confidence intervals are only drawn for the experiment without flow management. As can be seen, they are very small and this is also true for the other approaches. The approach without any flow management performs the worst in terms of playtime, having about 70 percent of the time a pre-playtime below 20 seconds. The Bandwidth-based and Round Robin approaches fare little better, but at the cost of a reduced video quality. The Round-Robin approach seems to outperform the Bandwidth-based approach, but this is because the video stalls earlier and so the Round Robin approach benefits much sooner from the reduced video size. Deep Packet Inspection and the Application-Awareness approach show by far the best performance with a pre-buffered playtime of 50 seconds and above for about 90 and 80 percent of the time, respectively. However, taking the conserved bandwidth and smaller resource overhead into account, the Application-Aware SDN approach appears to be the most viable.

V. Conclusion

In this paper we have shown the benefits of combining application-state information with SDN network control for network management for the example of YouTube streaming. We have seen that users can benefit profoundly from this approach compared to purely QoS-based methods. However, the improved performance comes at a cost of resource overhead. Generally, more signaling and computing is required to profit from the advantages in QoE. Currently, to achieve this, reactive flow setup with OpenFlow is required, which limits the applicability of this approach to smaller networks such as the scenario described in this paper. Still, with the ongoing trends towards Network Functions Virtualization and the increasing use of network processors, this may change in the future. Therefore, we aim to improve our approach by investigating more applications and their requirements as well as further network resource management strategies beyond path selection.

References

[1] M. Fiedler, K. Kilkki, and P. Reichl, “From quality of service to quality of experience,” in *Abstracts Collection of Dagstuhl Seminar*, vol. 9192, 2009.

[2] B. Staehle, M. Hirth, R. Pries, F. Wamser, and D. Staehle, “YoMo: A YouTube Application Comfort Monitoring Tool,” in *QoEMCS Workshop*, Tampere, Finland, Jun. 2010.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, 2008.

[4] A. R. Curtis, W. Kim, and P. Yalagandula, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1629–1637.

[5] S. Das, Y. Yiakoumis, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and P. D. Desai, “Application-aware aggregation and traffic engineering in a converged packet-circuit network,” in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference*. IEEE, 2011, pp. 1–3.

[6] M. Jarschel and R. Pries, “An OpenFlow-Based Energy-Efficient Data Center Approach,” *ACM SIGCOMM*, 2012.

[7] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang, “Understanding the impact of video quality on user engagement,” in *ACM SIGCOMM conference*. ACM, 2011, pp. 362–373.

[8] T. Hoßfeld, R. Schatz, M. Seufert, M. Hirth, T. Zinner, and P. Tran-Gia, “Quantification of YouTube QoE via Crowdsourcing,” in *IEEE (MQoE 2011)*, Dana Point, USA, 2011.

[9] Pica8, “Pronto 3290 OpenFlow Switch,” http://www.pica8.org/products/p3290.php.

[10] “Floodlight,” http://floodlight.openflowhub.org/, last accessed on 2013.03.21.

[11] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, “Iperf: The tcp/udp bandwidth measurement tool,” *htt p://dast. nlanr. net/Projects*, 2005.

[12] G. Combs *et al.*, “Wireshark,” *Web page: http://www. wireshark. org/last modified*, pp. 12–02, 2007.