SDN4Moodle: an SDN-based Toolset to Enhance QoS of Moodle Platform

Anderson H. da Silva Marcondes, Charles C. Miers, Maurcio A. Pillon, Guilherme P. Koslovski
Graduate Program in Applied Computing – Santa Catarina State University – Joinville – Brazil
anderson.marcondes@edu.udesc.br, {charles.miers, mauricio.pillon, guilherme.koslovski}@udesc.br

***Abstract*—Network configuration, performance, and congestion impact the Quality of Service (QoS) of Moodle Learning Management System (LMS). Moreover, LMSs composing modules have different QoS demands for each activity. Upon a network, they compete for resources with applications for social media, streaming, chat, cloud file synchronization, games, among others. Eventually, the network congestion can leave the LMS service unavailable for students and professors. Recently, the popularization of Software Defined Networks (SDN) revolutionized the network management. We propose SDN4Moodle, a toolset to increase the QoS of Moodle services. SDN4Moodle was implemented in conformity to Moodle's plugin guidelines, and it is controlled by internal Moodle events. Each action performed by a Moodle's user is dynamically translated into network forwarding rules, and switches configuration. Our experimental results indicates SDN4Moodle, an open-source system, efficiently achieves flows isolation and bandwidth control on shared networks.**

***Index Terms*—SDN, Moodle, QoS, OpenFlow**

I. Introduction

In a traditional network based on TCP/IP protocols stack, the development of traffic management polices is limited by the ossification of forwarding resources [1], and consequently the performance of network-intensive applications are impacted by TCP fairness goal [2], [3]. In other words, when n flows are sharing a network link, the protocol considers $1/n$ a fair fraction for each one [4]. However, in controlled environments (*e.g.*, clouds, academic, data centers) the fairness-driven approach is not preferable [5]. Eventually, the execution of some applications composes an operational bottleneck, such as data replication, servers' synchronization, or even the execution of educational activities (*e.g.*, exams, tutorials, streaming).

The academic routine is supported by LMS. LMS process is able to store, and disseminate, educational material to students and professors [6], [7]. Among the existing LMSs, we focus on Moodle open-source platform [8]. Approximately 87 thousand sites are officially registered on Moodle, distributed atop 232 countries, supporting up to 14 million courses and 118 million users [9], [10]. Acting as a content centralizer, the LMS Moodle is a potential bottleneck in the execution of educational activities. On a congested network scenario, the most noticeable problem observed by Moodle users is the slow loading of web pages while executing educational activities. Nowadays, with the ever-increasing number of multimedia applications disputing for networking resources, the eventual bandwidth reduction performed by the TCP protocol composes a performance barrier, impacting on throughput, and latency.

The application of QoS rules is a complex task [11] that commonly requires the administrator intervention to configure switches and routers. Even on infrastructures based on IntServ [12] or DiffServ [13] standards, QoS rules are composed up to layer 4, disregarding the application requirements and dynamic loads [14]. Although the SDN paradigm facilitated the network management by decoupling the control and data planes [11], the technical literature lacks on solutions to translate Moodle events into forwarding rules.

We claim specific forwarding rules must be prepared to enhance the performance of Moodle platform, as well as students and professors experience. These rules can be applied to each activity according to their demands. SDN is an elegant and contemporary way to support the claim. It is precisely in this opportunity our research hypothesis of this work was based. In short, this work proposes SDN4Moodle, a toolset to enhance QoS for Moodle platform. Our toolset considers administrator setting, activity characteristics and dynamic resources load to provider specific rules for each Moodle activity. SDN4Moodle is composed of a module (S4M_Monitor) in charge of collecting networking events for each user, and S4M_Agent, a module to store, process, and translate the events into forwarding rules. SDN4Moodle follows Moodle design guide and integrates to existing tools as optional plugins. In turn, SDN switches are configured based on the OpenFlow protocol [15], specifically, the metric tables are populated following user-generated events.

The remainder of this paper is organized as follows. Section II outlines challenges on enhancing QoS for the LMS Moodle and summarizes the potential application of SDN and OpenFlow. Related work is discussed on Section III. Section IV details the SDN4Moodle toolset, while Section V presents the experimental evaluation. Final considerations and perspectives are detailed in Section VI.

II. Literature Review and Motivation

A. LMS Moodle

Moodle is a web application, in which users interact based on predefined roles (*e.g.*, administrator, manager, professor, student). As summarized on Figure 1, users (front-end) access the core system through personal devices connected by network. The database is composed of 250+ tables used to store administrative and educational data. Focusing on systems interoperability [16], Moodle has a modular architecture, composed of a set of plugins, subsystems, and a kernel. This

organization allows the system customization according to institution requirements without changing its core components.

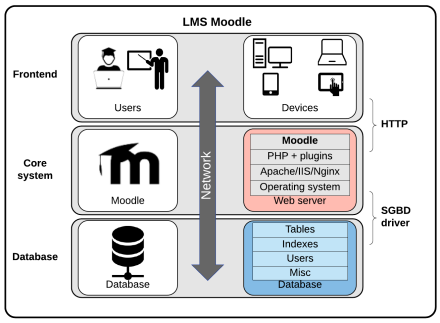


Fig. 1. Key elements composing the Moodle platform.

Events in Moodle are defined as atomic operations which describes something that happened during their use (users actions / administrative processes). In order to capture and process such data, each plugin must define which events must be monitored by implementing abstract classes. In the user's view, interaction with the Moodle platform is composed of activities. Users can do login, logout, exams, downloads, audio, and video streaming's. Activities have different characteristics and resource demands.

B. Network Requirements for LMS Moodle

In traditional networks, each equipment must be configured to be autonomous in decision making. One of the latent limitation of such approach is that devices are only able to interpret data from layer for which they were designed [17]. Information on source and destination ports, application, or sender environment are not commonly available and analyzed by traditional networks. Therefore, the enhancement of strategies driven by high-level details is practically unfeasible [15].

Moodle is highly dependent on networking infrastructure (internal components and users interaction). However, there is no default plugin to control the communication between LMS Moodle and network management systems. Natively, Moodle is unable to report its QoS demand for each specific event or user activity. All application traffic is treated as a single flow traveling the network, not allowing the prioritization of user activities or events.

C. Forwarding Moodle Data on SDN

The SDN has emerged as a flexible approach to program and manage computer networks, soften the complexity on configuration, operation and monitoring [18]. Considering the data plan is composed of packets which must be forwarded according to rules defined by the control plan, SDN advocates logical control can be implemented in a separated equipment, called controller. With the separation of data and control planes, there are two key aspects which enable the application of SDN to enhance QoS on Moodle. First and foremost, the flow-based forwarding abstraction enables the programmability of data plan based on standard Application Programming Interface (API). Secondly, the centralized management performed by a SDN controller provides full knowledge on network resources [19].

Natively at the southbound interface between the controller and switches (*e.g.*, OpenFlow [15]), the definition of forwarding criteria is limited to headers data from layers 2 to 4. In order to enable forwarding rules based on Moodle events, it is necessary to explore the northbound interface, *i.e.*, communication between application and controller. Thus, Moodle can notify the controller about the beginning and ending of specific events. A set of events defines a user activity, and its requirements (*e.g.*, burst, bandwidth reservation, tolerated latency). Thus, the controller can create specific QoS forwarding policies according to each activity.

III. Related Work

Literature comprises work which investigate the use of SDN to improve the performance of applications in varied contexts (*e.g.*, multimedia, high performance computing), distinct or joint acting on northbound and southbound interfaces.

Regarding exclusively to northbound, [20] developed SDN applications to optimize Hadoop clusters. Multimedia applications (*e.g.*, YouTube) are already able to improve their performance by using SDN [21]. The authors implemented a module to monitor the storage use of videos. This information is sent to a SDN controller in charge of prioritizing traffic when the buffer does not contain enough data to support the execution of a given video quality. Indeed, buffer size represents the quality experienced by a user. Following this line, [22] implemented a controller for multimedia delivery with end-to-end QoS. For the authors, the QoS requirements are translated as forwarding delay requirements, minimized through the monitoring of links utilization and reorganization of forwarding tables.

A joint approach is proposed by [23], defined as a new network paradigm, the Application Drive Network (ADN), which provides differentiated on-demand services for applications. The physical network is sliced into several isolated logical subnetworks, where each slice has its own topology and protocol to exclusively serve the hosted application. In turn, an extended SDN architecture was proposed by [19] to allow inspection of data beyond layer 4. Packets classified as table miss (*i.e.*, without corresponding entry in the switch flow-table) are intercepted between switches and controller. Latter, it is analyzed and the routing decision happens by querying an auxiliary table representing the application requirements. In addition, [24] proposed a network operating system for SDN guided by the QoS requirements reported during the Service Level Agreement (SLA) establishment.

In summary, related work indicate a real application's performance improvement on SDNs. However, none of the reviewed papers presented a proposal generic enough to support LMS Moodle. Therefore, the present work prioritizes the forwarding of Moodle's network traffic in a proactive way, based on internal events, following the ADN proposal [23]. Regarding the interface classification, the proposal from Section IV acts exclusively on northbound interface, without changing the OpenFlow protocol on southbound interface.

IV. SDN FOR MOODLE (SDN4MOODLE)

The execution scenario of SDN4Moodle is presented on Figure 2. Two users request three different activities ($A1$, $A2$, and $A3$) through SDN network. On Moodle server, the S4M_Monitor is responsible for monitoring internal events and for linking a set of events to users' activities. When interacting with Moodle, a set of user actions are translated into events, such as *(i)* start and end of the sessions; *(ii)* execution of exams and assignments; *(iii)* visualization of courses details; *(iv)* visualization of main dashboard; *(v)* download and upload; and *(vi)* video and audio streaming. Once user activity is identified, S4M_Agent builds specific rules and it reconfigures switches flow tables to enhance QoS for academic users. Then, each activity has specific QoS demand dynamically adjusted. $A1$ and $A2$, for instance, are requested by the same user but have distinct QoS requirements. On the other hand, $A1$ and $A3$ ask for the same configuration, but are requested by different users. In order to achieve this configuration, SDN4Moodle retrieves the network status from SDN controller, identifies the appropriated bandwidth configuration, and reconfigure all intermediate switches flow tables.

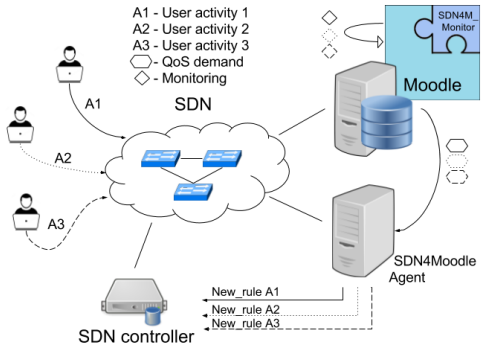


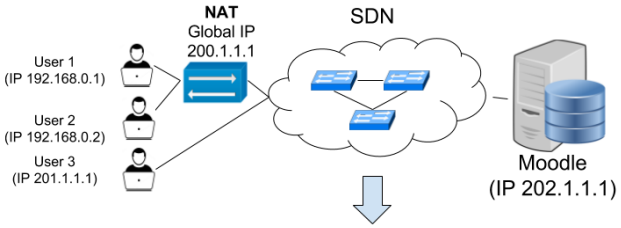
Fig. 2. Execution scenario and main modules of SDN4Moodle.

Motivated by adherence to available hardware, the OpenDayLight (ODL) controller (Lithium) was selected to support the SDN4Moodle. Specifically, using the ODL modules to discover resources and to enforce rules on switches. In turn, a private database (MySQL CS v.5.7.20) was developed to store events and switches configurations. A total of 7 tables stores the events generated by Moodle and toolset parameterization.

A. Identifying Network Flows from Moodle Activity

In order to identify the Moodle traffic on the SDN network, it is necessary to inspect the attributes available in the packets headers. As the specification of OpenFlow protocol is periodically improved, SDN4Moodle is based on its version 1.3 the most widely implemented on commercial devices. An OpenFlow packet is a tuple consisting of 12 components. Seven components are analyzed to identify the traffic generated to and from a Moodle activity: input switch port, IP protocol (IPv4), source/destination IP addresses, Ethernet type (2048), and TCP/UDP source/destination ports. As long as for each user activity it is necessary to register two flow-table entries on each switch (round trip traffic), the source/destination addresses consist of the user's device and the Moodle server on the way (upload to server), and the opposite in return.

Figure 3 shows an example of Moodle's flow modeling. For simplicity sake, each user has only one active activity request, although multiple activities are supported by SDN4Moodle.



User	Source address	Source port	Destination address	Destination port
User 1	200.1.1.1	50000	202.1.1.1	80
User 2	200.1.1.1	50001	202.1.1.1	80
User 3	201.1.1.1	50000	202.1.1.1	80

Fig. 3. Example of Moodle user flow identification.

Users 1 and 2 (Figure 3), using NAT, share the same IP address (200.1.1.1), Consequently, both addresses and source ports must be identified in order for the SDN controller to be able to enhance QoS. Due to the source address is the same for both users, the flows are differentiated by source port (50000, and 50001). Thus, a flow from Moodle User 1 is identified as tuple <200.1.1.1, 50000, 202.1.1.1, 80>, while for User 2 is denoted by tuple <200.1.1.1, 50001, 202.1.1.1, 80>. In addition, User 3 has a distinct IP address, and the flow is identified by <201.1.1.1, 50000, 202.1.1.1, 80>.

B. S4M_Monitor

In the absence of a plugin to capture and store events from official Moodle's repository, we developed one to carry out this task, storing data into Moodle and SDN4Moodle databases. When a user logs in Moodle, plugin notifies SDN4Moodle about events through database registers. Thus, SDN4Moodle sends commands to controller to create flow-table entries for all switches between user and Moodle server.

The development of S4M_Monitor was realized according to the guidelines for preparing Moodle plugins (version 3.0). This version of Moodle was chosen as the management API offers a robust solution to receive all events in a single monitoring service [25]. Being able to capture Moodle events, it becomes possible to create forwarding rules in network devices according to high-level requirements. Therefore, the application can notify the network of its demands at runtime, avoiding the network administrator from the configuration of temporary rules for congestion control. Table I summarizes events monitored by S4M_Monitor.

TABLE I
MOODLE EVENTS MONITORED BY SDN4MOODLE - S4M_MONITOR.

Event	Description
\core\event\user_loggedin	User login
\core\event\user_loggedout	User logoff
\core\event\course_viewed	Visualization of course homepage
\core\event\dashboard_viewed	Visualization of Moodle dashboard
\mod_page\event\course_module_viewed	Visualization of course page
\mod_quiz\event\course_module_viewed	Start of an exam
\mod_resource\event\course_module_viewed	Visualization o a multimedia file

C. S4M_Agent - QoS Control and Enhancement

S4M_Agent focuses on delivering events data from Moodle to OpenFlow controller. In addition, it also collects the counters available on each of OpenFlow switches to proactively calculate forwarding paths based on real-time data. Moreover, the network administrator can set additional parameters, *e.g.*, video characteristics (low, medium, and high network demand), Moodle authentication system, and OpenFlow drivers.

Each flow-table entry is formatted following the ODL API specification. In short, data sent to switches represent the combination of events database. For instance, to display a video, once the corresponding register is detected in database, a search is started to discover the required bandwidth configuration according to the configuration set defined by the administrator. The joint information is submitted to the switches by configuring flow and metric tables.

V. Experimental Evaluation

A. Method

As main metric for discussion, we analyze the background bandwidth consumption. The objective is to show that bandwidth reservations enforced by SDN4Moodle decreases the slice of network resources available to other applications, on benefit of the Moodle users. We ingested the background traffic using iperf v. 2.0 application, configured with bidirectional traffic. Iperf provides bandwidth consumption reports, so we inferred Moodle's consumption by subtracting it from total network bandwidth. Regarding to student navigation profile, a set of activities (*i.e.,* text, audio, and video) were selected to represent a session. The text activity is a multiple-choice exam, 10 questions with 4 alternatives each, being a question presented at a time. Audio and video streaming's' are directly displayed in Moodle media player without full buffering in user's device. Video file has 10 minutes length (MP4 format), and 3 types of resolution quality were selected: 720p, 480p, and 240p. The audio file was extracted from the video one, in MP3 format, with 192 Kbps. Table II resumes the sequence of user steps, and monitoring tasks.

TABLE II
Steps for data monitoring executed for each Moodle's user.

Step	Task
A	Start monitoring Moodle's user network interface
B	Start iperf servers and clients
C	Moodle login
D	Start text event
E	End text event
F	Start audio activity
G	End audio activity
H	Start low resolution video streaming
I	End low resolution video streaming
J	Start medium resolution video streaming
K	End medium resolution video streaming
L	Start high resolution video streaming
M	End high resolution video streaming
N	Logoff from Moodle
O	End monitoring Moodle's user network interface
P	End iperf servers and clients

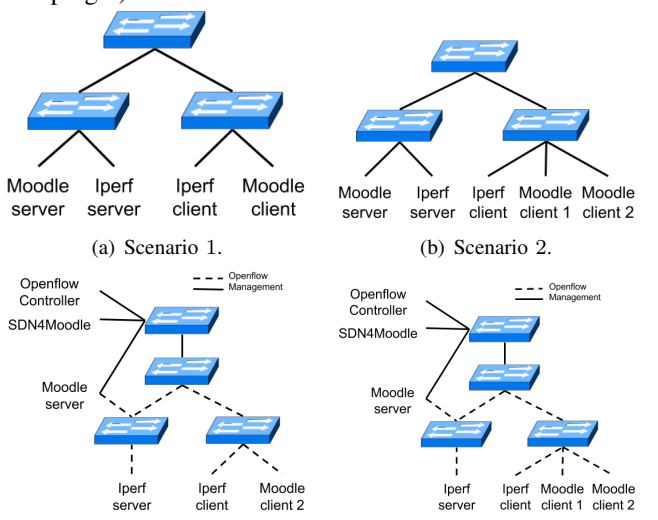
Initially (Table II), the user's network interface is monitored (A). From this moment, only Moodle traffic is received and sent over the network interface. Latter, iperf server and clients are initialized to ingest the background traffic (B). User interaction starts on access to login page (C). Upon authentication, user is taken to list of available courses, selecting the textual exam (D), and sending responses to server after answering the last one (E). The user then returns to the course topics.

Next step is to run audio streaming (F), and latter return to course topic list (representing the end of audio activity) (G). The same script is followed for displaying videos (H, J, and L for starting / I, K, and M for ending). Upon completion of activities, the user logs off (N), ending their interaction with Moodle. Finally, network interfaces monitoring is stopped (O), as well as the iperf process (P). For each of these actions, log files are generated and later used to compose the analysis.

Due to the limited processing capacity of virtualized switches [26], maximum bandwidth was set at 25 Mbps. Values above this threshold implies on packet loss. Thus, bandwidth reservation was performed as follows: 2 Mbps after login; 4 Mbps for textual exams, audio, and low quality videos; 8 Mbps and 16 Mbps for medium and high quality videos, respectively.

B. Experimental Scenarios

Four experimental scenarios were defined and are presented on Figure 4. SDN4Moodle communicates with both OpenFlow controller (sending actions to create, update and remove flow-table entries) and Moodle server (receiving events data from LMS plugin).



(a) Scenario 1. (b) Scenario 2.

(c) Scenario 3. (d) Scenario 4.

Fig. 4. Experimental topologies.

Tables III and IV shows the hosts and switches configuration, respectively.

TABLE III
Hosts configurations.

Host	CPU	RAM
Moodle server	AMD Athlon TF-20 1.6 GHz	3.6 GB
Controller	AMD Athlon TF-20 1.6 GHz	3.6 GB
SDN4Moodle	Intel Core I5 3340m 2.7 GHz	8 GB
Iperf server	Intel Atom N2701.6 GHz	1.5 GB
Iperf client	Intel Atom N2701.6 GHz	1.5 GB
Moodle user 1	Intel Pentium T43002.1 GHz	3 GB
Moodle user 2	Intel Pentium T43002.1 GHz	3 GB

TABLE IV
SWITCHES CONFIGURATIONS.

Switch	Model	Interfaces	OpenFlow
Root	TP-Link TL-WR1043ND	1 Gbps	OpenFlow 1.3
Edge 1	Generic 8 interfaces	100 Mbps	No
Edge 2	Generic 8 interfaces	100 Mbps	No
Management	Linksys SE 1500	1 Gbps	No

GNU/Linux Ubuntu 16.04 64bits was used for all hosts, while OpenFlow switches were configured with OpenWRT and CPqD user-space implementation [27].

1) Scenario 1: First scenario, depicted by Figure 4(a) comprises of 4 servers interconnected by 3 switches without the execution of SDN4Moodle. On this baseline scenario, network topology is saturated by background traffic while one Moodle's user access the system and executes a set of predefined actions (from Table II). These results are used as a baseline threshold on the comparison with Scenario 3.

2) Scenario 2: The testbed comprises 3 switches and 5 servers (Figure 4(b)) without executing SDN4Moodle. Different from Scenario 1, two Moodle's users access the system and share the network with background traffic. Results are used as baseline for comparison with Scenario 4.

3) Scenario 3: SDN4Moodle is enabled to manage the topology depicted by Figure 4(c). In order to represent a hybrid scenario, OpenFlow is enabled on root switch understanding the QoS rules defined by SDN4Moodle, while it is disabled on edge switches. It is worthwhile to mention the existence of a management network, parallel to the experimental SDN. Thus, OpenFlow-aware traffic is denoted by dashed lines while management data is represented by solid lines.

4) Scenario 4: Scenario 3 is extended by adding a new Moodle's user, composing the testbed depicted by Figure 4(d).

C. Results and Discussion

Figures 5, 6, 7, and 8 summarize the bandwidth consumption of background traffic for all scenarios. Each event occurrence is marked by a dashed vertical line. Events triggered before/after user's interaction were omitted (A, B, O, and P).

1) Scenario 1: Initially, Figure 5 shows the background traffic consumption without QoS enforcement performed by SDN4Moodle (topology from Figure 4(a)). The network is fully occupied by TCP iperf sessions when Moodle's user is not actively interacting with the system (events C, and D).

Fig. 5. Scenario 1: background traffic (1 user) without SDN4Moodle control.

Consequently, Moodle's user perceives delayed responses from server. The decrease on background bandwidth consumption on events (F, H, J, and L) is attributed to fair sharing performed by TCP. However, even sharing the bandwidth, Moodle's user has no guarantee on minimum configuration. It is worthwhile to mention, actions translated into short flows (a few packets at most) [28] the reaction imposed by TCP congestion control algorithm may not be perceived (*e.g.*, login, logoff, textual exam).

2) Scenario 2: Two Moodle's users are simultaneously sharing and disputing the network resources with background traffic, without executing SDN4Moodle for controlling the QoS (Figure 4(b)). On Figure 6, the maximum network throughput (25 Mbps) was achieved, summing up users traffic and background TCP flows, mainly on events with burst communication between Moodle's users and server. Eventually, outliers are perceived due to the periodicity of data monitoring (1 s). As expected, this second baseline indicates long term background TCP flows can dominate the network throughput disregarding the Moodle's usage.

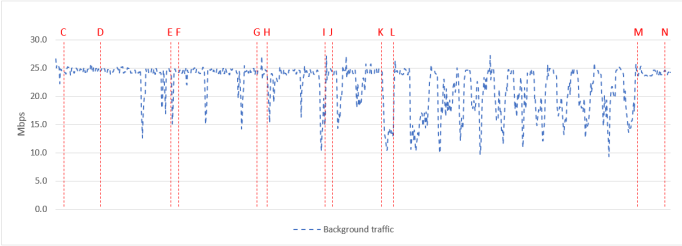


Fig. 6. Scenario 2: background traffic (2 users) without SDN4Moodle control.

3) Scenario 3: Using SDN4Moodle to actively enforce the QoS rules, background traffic is controlled on testbed depicted by Figure 4(c). Figure 7 compares the traditional network profile (from Scenario 1, blue line) with SDN4Moodle enforcement (red line). First observation is the sharp drop on events in which it is necessary to update the Moodle's user rules (mainly in events C, D, E, F, H, I, K, L, and N). This happens due to the existing limitation on OpenFlow protocol installed on root switch, which excludes the packet queue when the metric table is updated. Finally, the comparison shows that SDN4Moodle accurately performs the bandwidth reservation, causing the secondary traffic consumption to vary according to the reservations performed.

Fig. 7. Scenario 3: background traffic (1 user) with and without SDN4Moodle control.

4) Scenario 4: Figure 8 shows the results with 2 Moodle users executing on testbed depicted by Figure 4(d)). It is possible to note as the users execute their activities, the greater is the decrease of secondary traffic flow. On low-resolution video presentation (240p, H-I intervals), maximum bandwidth reached by background traffic was about 17 Mbps, which

corresponds to the total network bandwidth (25 Mbps) minus the total allocated for Moodle users (4 Mbps for each). On medium-resolution video presentation (480p, J-K intervals), maximum secondary traffic was about 9 Mbps, which also matches bandwidth reservation rules (Section V-A), being 25 Mbps less the 8 Mbps for each user. On high-resolution video presentation (720p, L-M interval), residual bandwidth available to secondary traffic would be negative (-7 Mbps), or in other words, causing the allocation of entire network congested link to Moodle users, in detriment of the secondary traffic. Finally, as the main difference between Scenarios 2 and 4, a lowest transfer time of the high-resolution video is perceived. It is also noticed that not all the available bandwidth was used, since the sum of the rates of both users was only about 56% of the total available.

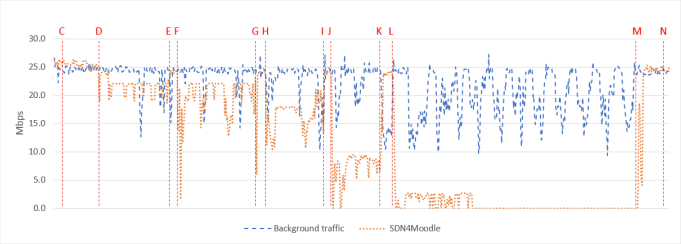


Fig. 8. Scenario 4: Background traffic (2 users) with and without SDN4Moodle control.

VI. Considerations & Future work

We presented SDN4Moodle, a proposal to enhance QoS for Moodle on SDN-based networks. The QoS enforcement is performed by reserving bandwidth in execution time, based on events informed by the LMS. A prototype was implemented, in order to analyze the feasibility of SDN4Moodle. Two software modules were developed: a Moodle plugin for collecting the users events, and a tool for the configuration of flow-table and metric-table entries in forwarding switches. The prototype was evaluated in an SDN topology with real physical and virtualized devices. Each Moodle's user performed typical academic activities (*e.g.*, login, exams, audio, and video streaming) while SDN4Moodle enhance experienced QoS by allocating predefined bandwidth rules. Our results show not only the feasibility of SDN4Moodle, but how it is efficient on the presented scenarios. As future work we highlight the analysis with robust switches to analyze the scalability of SDN4Moodle, as well as the development on real academic network.

Acknowledgments This research was partially supported by the UDESC and FAPESC, and was developed at the LabP2D.

References

[1] M. Handley, "Why the Internet only just works," *BT Technology Journal*, vol. 24, pp. 119–129, July 2006.

[2] V. G. Cerf and R. E. Icahn, "A protocol for packet network intercommunication," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 71–82, 2005.

[3] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi, "Proportional rate reduction for tcp," in *Proc. of ACM SIGCOMM Conference on Internet Measurement 2011, Berlin, Germany*, 2011.

[4] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[5] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: Sharing the network in cloud computing," in *Proc. of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, pp. 22:1–22:6.

[6] T. J. McGill and J. E. Klobas, "A task–technology fit view of learning management system impact," *Computers & Education*, vol. 52, no. 2, pp. 496–508, 2009.

[7] G. D. Bailey, "Wanted: A road map for understanding integrated learning systems." *Educational Technology*, vol. 32, no. 9, pp. 3–9, 1993.

[8] M. Dougiamas and P. Taylor, "Moodle: Using learning communities to create an open source course management system," in *Proc. of the EdMedia: World Conf. on Educational Media and Technology, Honolulu, Hawaii*, 2003.

[9] Moodle. (2017) Moodle statistics. [Online]. Available: https://moodle.net/stats/

[10] M. Ş. Kuran, J. M. Pedersen, and R. Elsner, "Learning management systems on blended learning courses: An experience-based observation," in *International Conference on Image Processing and Communications*. Springer, 2017, pp. 141–148.

[11] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[12] R. Braden, D. Clark, and S. Shenker, "RFC 1633 integrated services in the internet architecture: An overview," RFC 1633, 1994.

[13] D. Grossman, "New terminology and clarifications for diffserv," Internet Requests for Comments, RFC Editor, RFC 3260, April 2002.

[14] J. Babiarz, K. Chan, and F. Baker, "RFC 4594: Configuration Guidelines for DiffServ Service Classes," IETF, Tech. Rep., 2006.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[16] A. Brown and G. Wilson, *The architecture of open source applications, volume ii*. Kristian Hermansen, 2012, vol. 2.

[17] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, 1st ed. Addison-Wesley Professional, 2015.

[18] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "Policycop: an autonomic qos policy enforcement framework for software defined networks," in *SDN for Future Networks and Services (SDN4FNS),*. IEEE, 2013, pp. 1–7.

[19] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman, "Application-aware data plane processing in sdn," in *Proc. of the Third Workshop on Hot Topics in SDN*. ACM, 2014, pp. 13–18.

[20] S. Zhao, A. Sydney, and D. Medhi, "Building application-aware network environments using sdn for optimizing hadoop applications," in *Proc. of SIGCOMM 2016*. ACM, 2016, pp. 583–584.

[21] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "Sdn-based application-aware networking on the example of youtube video streaming," in *2013 Second European Workshop on Software Defined Networks*, Oct 2013, pp. 87–92.

[22] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Signal Inf. Proc. Association Annual Summit and Conf. (APSIPA ASC)*, 2012, pp. 1–8.

[23] Y. Wang, D. Lin, C. Li, J. Zhang, P. Liu, C. Hu, and G. Zhang, "Application driven network: Providing on-demand services for applications," in *Proc. of ACM SIGCOMM*. ACM, 2016, pp. 617–618.

[24] K. Jeong, J. Kim, and Y.-T. Kim, "Qos-aware network operating system for software defined networking with generalized openflows," in *Network Operations and Management Symposium*. IEEE, 2012, pp. 1167–1174.

[25] Moodle. (2017) Moodle event 2. [Online]. Available: https://docs.moodle.org/dev/Event_2

[26] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer." in *Hotnets*, 2009.

[27] CPqD, "OpenFlow 1.3 for OpenWRT," dec 2017, original-date: 2012-07-23T13:08:14Z. [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[28] M. Noormohammadpour and C. S. Raghavendra, "Datacenter Traffic Control: Understanding Techniques and Trade-offs," *IEEE Communications Surveys & Tutorials*, dec 2017.