Designing and Prototyping of SDN Switch for Application-Driven Approach

Diego Nunes Molinos, Romerson Deiny Oliveira, Marcelo Silva Freitas, Natal Vieira de Souza Neto, Marcelo Barros de Almeida, Flávio de Oliveira Silva and Pedro Frosi Rosa

Abstract Currently, the Internet has become a limiting factor for its evolution. Applications are being developed from a new perspective of network utilization, demanding more Quality of Service (QoS) and Quality of User Experience (QoE). Approaches that aim to redesign the architecture, for example, Software Defined Networks (SDN), have become popular in the field of computer networks in an attempt to minimize the problems experienced by TCP/IP. In theory, SDN Networks naturally leave all the flexibility and programmability of the network in charge of the control plane, disregarding the data plane's ability to improve the QoS and QoE for users. This work presents the specification and the development of a prototype

Diego Nunes Molinos
Faculty of Computing, Federal University of Uberlandia (UFU), Joao Naves de Avila Avenue, 2121, Santa Monica, Uberlandia, Brazil, e-mail: diego.molinos@ufu.br

Romerson Deiny Oliveira
High Performance Networks Group, University of Bristol, Woodland Road, Bristol, United Kingdom, e-mail: romerson.oliveira@bristol.ac.uk

Marcelo Silva Freitas
Department of Exact Sciences, Federal University of Jataí (UFJ), Jataí, Brazil, e-mail: msfreitas@ufg.br

Natal Vieira de Souza Neto
Faculty of Computing, Federal University of Uberlandia (UFU), Joao Naves de Avila Avenue, 2121, Santa Monica, Uberlandia, Brazil, e-mail: natalneto@ufu.br

Marcelo Barros de Almeida
Electrical Engineering Faculty (FEELT), Federal University of Uberlandia (UFU), Joao Naves de Avila Avenue, 2121, Santa Monica, Uberlandia, Brazil, e-mail: marcelo.barros@ufu.br

Flavio de Oliveira Silva
Faculty of Computing, Federal University of Uberlandia (UFU), Joao Naves de Avila Avenue, 2121, Santa Monica, Uberlandia, Brazil, e-mail: flavio@ufu.br

Pedro Frosi Rosa
Faculty of Computing, Federal University of Uberlandia (UFU), Joao Naves de Avila Avenue, 2121, Santa Monica, Uberlandia, Brazil, e-mail: pfrosi@ufu.br

of *Switch* with *MAC* driven by the applications for SDN networks. Although it is possible to reconfigure the behavior of the network element, the *MAC* remains the same. Compared to other similar approaches, this proposal can expose, through the fine-grained, the low-level forwarding rules logic to the control plane through an orchestrator module, systematically allowing reprogramming. We carried out a case study using the Entity Title Architecture (ETArch) to show the ability of *Switch* to handle parameters, such as priority and bandwidth, in real-time.

1 Introduction

Internet applications are demanding more and more resources from the network. Although those applications require mobility, security, energy efficiency, bandwidth, QoS, and QoE, they usually leave to the network the best scenario to satisfy their requirements. Therefore, SDN networks have offered better and more efficient control over networking devices from a logically centralized controller to improve network management.

Due to the decoupling of the data plane and control plane in SDN, a distance from the functions implemented in hardware and software was created, making the infrastructure level a single granular network fabric. Furthermore, it contributed so that all flexibility and programmability were carried over to the control plane, disregarding the data plane's ability to improve the QoS and QoE for users.

Some proposals aim to explore the flexibility of the network through software solutions, for example, [15] and [9]. However, these solutions typically assume link-level connectivity and bypass data plane resources to provide QoS and QoE. In general, SDN implementations use OpenFlow-based switches or some virtualization technique. Although it is possible to reconfigure the behavior of the network element, these actions do not offer flexibility, making it challenging to manipulate QoS and QoE parameters at the *MAC* level.

This work presents a new switch for SDN networks with *MAC* driven by the application. This prototype is Linux-based, and it enables flexibility and programmability in the network's data plane. With our approach, the *MAC* remains the same, offering backward compatibility, and we go further to provide control to QoS and QoE parameters that impact the *MAC* decisions. In addition, we present a case study using the Entity Title Architecture (ETArch) to demonstrate and validate our switch. ETArch uses the SDN paradigm and is an architecture with features and support to context-oriented algorithms that allow adjustments according to the application requirements [2]. With our work, ETArch can modify the switches parameters changing their operation's context to guarantee the QoS and QoE driven by the application [2].

The remainder of the document is structured as follows: Section 2 presents the overview of flexibility in the SDN Data Plane and the background of the ETArch. Section 3 offers Switch prototype design details. Section 4 provides the results, and finally, Section 5 presents the conclusion and future work.

2 Background

This section provides a background about the programmability and flexibility in the SDN field and a brief about ETArch architecture.

2.1 Programmability and flexibility in SDN Data Plane

SDN-based networks present an innovative way of rethinking computer networks in the Future Internet field. According to [13], this paradigm aims to create a well-defined logical interface for network control, abstracting the network and providing reconfigurability of services at runtime. Its uniqueness lies in the fact that it gives programmability for the network by decoupling the control and data planes in the architectural design of the network.

Since the beginning of SDN, there has been an adhesion to use the OpenFlow technology. According to [4], OpenFlow enables the creation of configurable flows from a central software-based controller. It separates the data plane from the control plane and presents a centralized programming model to manage the network elements. It offers an easy way to develop an L2 solution to support SDN scenarios through the abstraction of the network infrastructure level.

The interface between the centralized control plane and the forwarding elements is established by the OpenFlow protocol [4] [15]. It is an out-of-band interface in which the controllers are connected to the network devices through physically dedicated links used exclusively for control traffic [7]. Fig. 1 illustrates OpenFlow from the point of view of communication interfaces with details of the data plane.

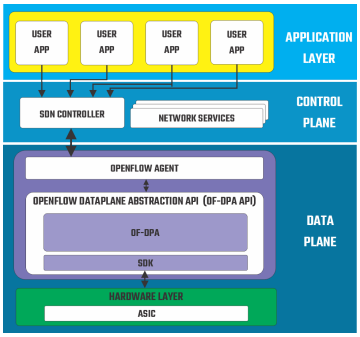


Fig. 1 OenFlow architecture interface. Figure adapted from [10]

As shown in Fig. 1, there is a dependency on using OpenFlow on legacy hardware platforms, making it difficult to improve any low-level network features [6]. From the point of view of actions implemented by flow, the OpenFlow has a coarse-grained concerning managing flows. There is an entry of flows and one action as-

sociated with each entry, and the action is applied to the entire flow. There is no treatment for each primitive but a limited set of actions per stream.

The drawback observed in the OpenFlow-based switches is that it requires, before startup, a manual configuration of your controller's IP address and TCP port [7]. Besides, it is necessary to evolve OpenFlow, expanding the range of equipment, protocols, content headers to support new network architectures, and this requires continuous modification in its specifications [8].

Solutions OpenFlow-based make it challenging to implement new features at the network's low level. Their policies are applied to the entires flows and not per primitives. In addition, the low level of OpenFlow switch the *MAC* remains the same.

In response to OpenFlow difficulties, the Programming Protocol-Independent Packet Processors (P4) [14] emerged, a high-level programming language for programming network devices in the data plane. The drawback in using P4 is that the solution requires a translation infrastructure to be implemented down to the processor level. In addition, using high-level synthesis tools to generate custom hardware can be an approach that introduces additional complexity into the design, with results that are still not so satisfactory.

2.2 Programmability and flexibility in ETArch Data Plane

The ETArch has been designed to approximate the Application layer semantically to the lower layers, allowing the requirements defined by the applications to permeate through the architecture layers. ETArch is based on SDN and presents innovations related to addressing network elements by the separation of location and identification, which intrinsically enables multicast transmissions and network mobility [1].

From the ETArch perspective, it's essential to comprehend the concepts of entity, Workspace, and the control agent to understand how the architecture works.

Entities are elements that want to communicate in a distributed environment [2]. It can be applications, devices, computers, network elements, etc. Entities are only identified by a title and a set of capabilities.

The Workspace represents an instance of communication driven by the application and carries a set of capabilities of the communication domain. As an example of the application requirements, can be mentioned the VoIP applications, where the transmission delays can be smaller than 150*ms*. However, delays above 400*ms* can impair transmission. Therefore, the Workspace created to serve a VoIP communication must operate with a transmission delay between 150*ms* and 400*ms*. For it to happen, all the architecture layers must be able to understand the application's requirements [2].

2.2.1 Domain Title Service - DTS

A Domain Title Service (DTS) represents a set of ETArch controllers in a distributed system. The goal of DTS agents is to separate the network into manageable subnets within ETArch [2]. A Domain Title Service Agent (DTSA) acts in the control plane, managing the network elements. In addition, DTSA is responsible for creating, managing, and dropping Workspaces on network elements. A typical ETArch environment is illustrated in Fig. 2.

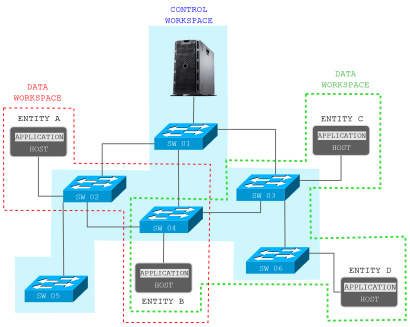


Fig. 2 Typical ETArch Enviroment with Data Workspace, Control Workspace, DTSA and entities. Figure from [11].

As shown in Fig. 2, DTSAs act on the network control plane and all communication between DTSA's and network elements is performed through a Control Workspace. In turn, the network elements act on the network data plane level, allowing entities to communicate within a communication domain, a Data Workspace.

3 Designing and Prototyping of SDN Switch Application-Driven Approach

This section will be present the design and prototyping of SDN Switch.

3.1 ETSCP Protocol

ETArch Switch Control Protocol (ETSCP) was designed by [11] to manage the control communication between DTSA and Switches in the ETArch network. There is no application entity involved in this control communication, which means, applications do not use ETSCP. The ETSCP format is similar to Ethernet protocol (802.3 [12]) to maintain compatibility with the current network interface cards, allowing the reuse of available technology. The changes in the content of the Ethernet header are in the source *MAC* address and destination *MAC* addresses field to support the

identification of the *Title (Workspace)* + *Source Entity* and the control message type added in the payload. Fig. 3 illustrates the message format used by ETSCP.

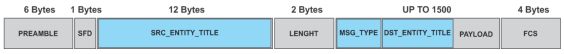


Fig. 3 ETSCP message format

About Services and Vocabularies, the ETSCP messages are *Add Workspace*, *Edit Workspace*, *Update Workspace* and *Remove Workspace*. All the ETSCP services are connectionless with acknowledgments, and ETSCP messages follow the taxonomy of confirmed services, Request/Indication, and Response/Confirmation. A new service in the ETSCP that acts on the scheduling policies to ensure QoS through non-ETArch communication was instituted, the Update Regular Connection service. Table 1 offers a clear association between the protocol services and their messages.

Table 1: ETSCP Services and Vocabulary

Service	**Messages**	**Functional Description**
Create	*AddWkpREQ*	Confirmed service.
Workspace	*AddWkpRESP*	To create new Workspaces in the switch.
Edit	*EditWkpREQ*	Confirmed service.
Workspace	*EditWkpRESP*	To modify parameters of the Workspace.
Remove	*RemWkpREQ*	Confirmed service.
Workspace	*RemWkpRESP*	To remove a Workspace in the Switch.
Update	*UpdateWkpREQ*	Confirmed service.
QoS Parameters	*UpdateWkpRESP*	To modify the QoS parameters of the switch's schedulers.
Regular Connection	*UpdateRegConREQ*	Confirmed service.
QoS Parameters	*UpdateRegConRESP*	To modify the QoS parameters of the switch's schedulers.

3.2 SDN Switch FSM - Finite State Machine

Modeling a solution means building processes capable of interpreting messages and acting according to the context. The design of the SDN Switch was specified using a finite state machine (FSM). The diagram in Fig. 4 formally describes the behavior of the services defined by the ETSCP protocol.

The state ACTIVE is the initial and final state. All the other states are signalized in different colors according to each service. The state PARSER is responsible for analyzing all the messages and forwarding them to the other states according to each service. In the sequence are presented a brief description of the modeled services.

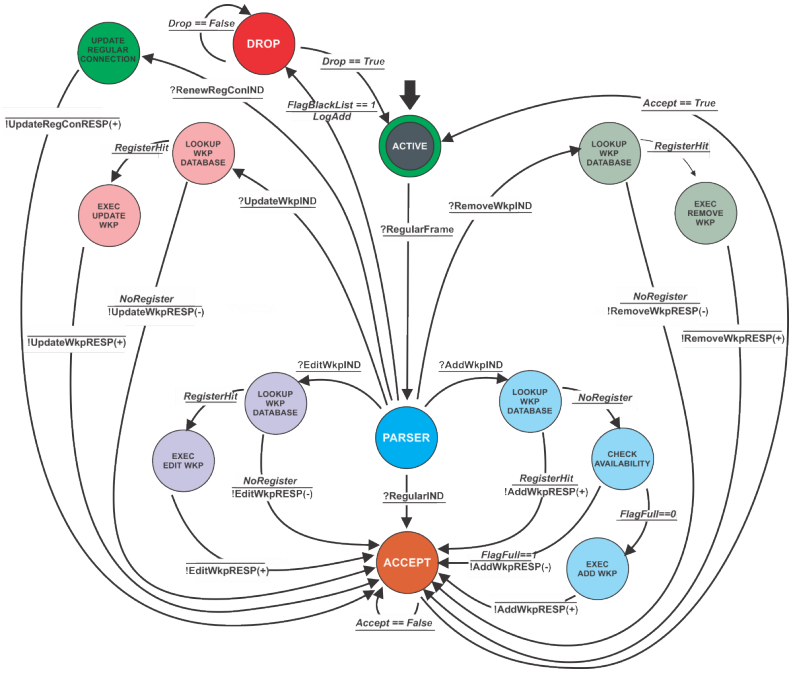


Fig. 4: Switch SDN - Finite State Machine (FSM)

1. *Create Workspace:* add support to the new schedulers' policies when the DTSA needs to create a control logic domain to respond to an entity request to create a data Workspace. When DTSA wants to create a Workspace, it sends an *AddWkpREQ* message to SDN Switch (*AddWkpIND*). Then, it verifies the existence of Workspace on the *WorkspaceDataBase* and checks the Switch availability to support this new Workspace, whether both of those actions before were true the Workspace will be created, and an ACK is sent or a NACK if not.
2. *Edit Workspace:* attach or remove a *Entity* in the existing Workspace, extending the Workspace's domain to reach out to the new *Entity* and update the scheduler's policies. When DTSA needs to edit a Workspace, it sends an *EditWkpREQ* message to SDN Switch (*EditWkpIND*). It verifies the existence of Workspace on the *WorkspaceDataBase*, the Workspace will be edited, and an ACK is sent or a NACK if not.
3. *Update Workspace:* change the QoS parameters of the existing Workspace. The QoS parameters used for schedulers policies are *Priority* that define the frame's prioritization per port, *Rate and Ceil* define the assigned bandwidth to each Workspace, *Burst and CBurst* determine the size, in bytes, to control the duration of the bursts.
4. *Remove Workspace:* remove the existing Workspace from the SDN Switch. When DTSA needs to remove a Workspace, it sends a *RemoveWkpREQ* message to

SDN Switch (*RemoveWkpIND*). Then, it verifies the existence of Workspace on the *WorkspaceDataBase*, whether it was true, the Workspace will be removed, and an ACK is sent or a NACK if not.

5. *Update Regular Connection:* change the QoS parameters of the existing regular traffic. The QoS parameters used for schedulers policies are the same. Any frame that is not a control frame or an ETArch frame will match the general forwarding rule.

3.3 Architecture of SDN Switch

From the point of granularity, the SDN Switch proposed has a fine granularity concerning the legacy switches. It allows forwarding policies oriented to frame and not the physical or logical ports as legacy equipment. Figure 5 present a view of the architecture and organization of SDN Switch.

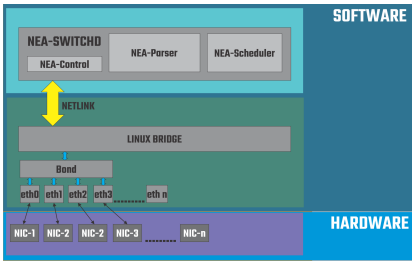


Fig. 5 SDN Switch Architecture and Organization

The Switch proposed in this work is a Linux-Based, the use Bridge module, support for 802.1 standards including 802.1P/Q and VLANs, and managing multiple network interfaces (NIC) justify the adoption of the Linux operating system. Furthermore, Linux Kernel natively supports several L2/L3 network functions, and the Bridge and Traffic Control modules are essential to promote traffic aggregation and manipulate forwarding traffic.

3.4 NEA-SWITCHD Module

This module is responsible for orchestrating all the elements in the SDN Switch, aiming to offer all the SDN Switch programmability and flexibility. The NEA SWITCHD module comprises NEA-Control, NEA-Parser, and NEA-Scheduler. The entire SDN Switch architecture operates sequentially. All primitives are queued and handled one by one by the NEA-SWITCHD module. Figure 6 present a view of the architecture and organization of the NEA SWITCHD module divided into software (Kernel Space and User Space) and hardware layer.

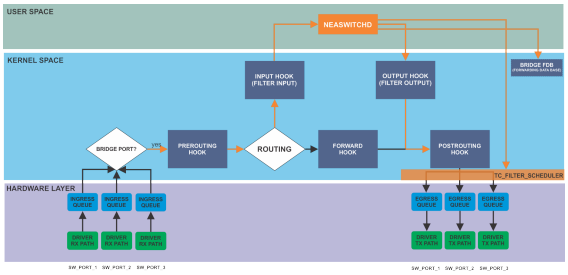


Fig. 6 NEA SWITCHD module Architecture and Organization

The NEA SWITCHD module interacts with Linux Kernel. In general terms, this module analyzes all frames that pass through the Bridge, schedules them, and orchestrates the policies within the scheduler defined by the network controller.

4 Tests and Prototyping Experimentation

The test aims to validate the Switch's ability to modify its control parameters to ensure the QoS of the applications. As depicted by Fig. 7, sequence diagrams show when messages are sent and received between Entities, Switch, and DTSA. For Workspace creation or edition, an entity sends the first message, and it is a data message addressed to a random Workspace. When Switch receives it, there is no forwarding policy for this message addressed, and the controller is triggered to configure or edit forwarding rules on the Switch. After that, DTSA sends a `AddWkp_Req` or `EditWkp_Req` to configure the Switch.

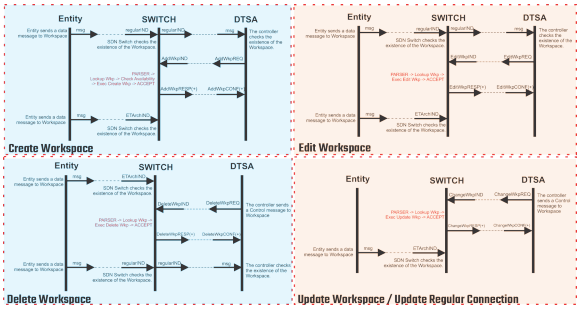


Fig. 7 Message Exchange to Service Executions. All message exchange was performed by a lightweight application that sends frames directly from the L2 layer. This application was written using the Python language and institutes ETSCP type messages.

Regarding removing Workspaces and updating QoS parameters, it begins with the DTSA. DTSA sends a `RemoveWkp_Req`, `RenewReg_Con_Req`, or `UpdateWkp_Req` to reconfigure the Switch, and after this, the Workspace is completely removed from Switch, or its QoS parameters are updated.

Fig. 8 illustrates the environment developed for the verification of the SDN Switch's ability to configure and reconfigure the application-oriented QoS parame-

ters. The ETSCP control messages are sent to Switch from the lightweight version of the DTSA. In addition to DTSA, there are two entities attached in a Workspace, and one of them runs a stream application.

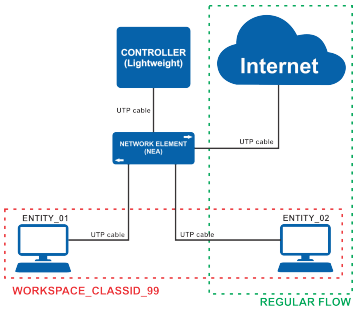
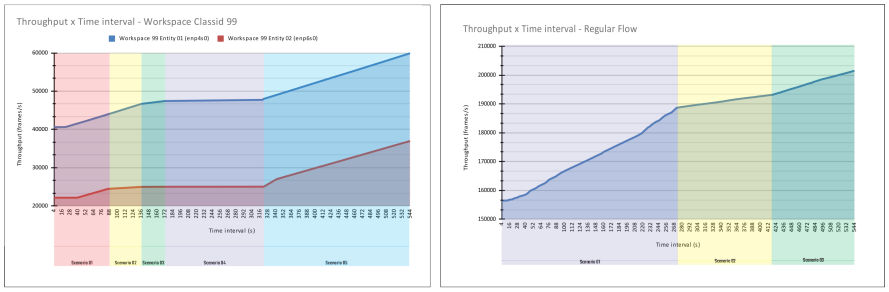


Fig. 8 Scenario to validate the SDN Switch performance concerning the control messages sent by the DTSA. The Entities are applications written in Python that exchange ETArch messages. At the same time, Entity 02 has been running an application streaming.

The metrics observed are (i) the ability to understand control messages from DTSA and (ii) the ability to configure application-oriented QoS parameters. Furthermore, the DTSA will act on the Workspace and regular flow to adjust the throughput through the QoS parameters.

In Workspace communication, the Entity_01 sends frames with a payload size of 58 bytes destined for the Workspace_Classid_99. Parallelly, the Entity_01 sends frames with a payload size of 1253 bytes with the target workspace_Classid _99. Although the frame sizes are different, stricter QoS policies allow less traffic in bytes by Workspace, and less strict QoS policies allow more traffic through Workspace.



(a) (b)

Fig. 9: Throughput (f/s) x Time interval (s) - Workspace and Application Streaming.

The SDN Switch throughput related to the Workspace during the time interval of 544 seconds is shown in Fig 9 (a), the QoS parameters are changed five times (Scenarios 1, 2, 3, 4, and 5), and in Fig. 9 (b), the QoS parameters were modified three times (Scenario 1, 2, and 3). A clear association between the Traffic rate and QoS parameters configuration about Workspace communication and Application streaming is observed in Table 2. It is essential to point out that during the tests, the *Burst*

and *Churst* parameters were not changed, just the priority and available bandwidth do. Both scenarios (a) and (b) presented in Figure 9 were evaluated concomitantly using the same time interval, and all the results were collected in real time.

Table 2: Traffic rate and QoS parameters configuration

Entity/Worskpace Communication	**Traffic rate (f/s)**	**QoS parameters**
Entity_01	*26,91 frames/s*	Scenario 01
Entity_02	*39,11 frames/s*	Priority.: 2, Rate: 100Mbit, Ceil: 100Mbit.
Entity_01	*10,04 frames/s*	Scenario 02
Entity_02	*53,79 frames/s*	Priority.: 7, Rate: 100Kbit, Ceil: 100Kbit.
Entity_01	*1,0 frames/s*	Scenario 03
Entity_02	*10,4 frames/s*	Priority: 7, Rate: 10Kbit, Ceil: 10Kbit.
Entity_01	*0,10 frames/s*	Scenario 04
Entity_02	*2,10 frames/s*	Priority: 7, Rate: 1Kbit, Ceil: 1Kbit.
Entity_01	*53,73 frames/s*	Scenario 05
Entity_02	*56,70 frames/s*	Priority: 2, Rate: 1Mbit, Ceil: 1Mbit.
Entity/Application Streaming	**Traffic rate (f/s)**	**QoS parameters**
Entity_02	*119,86 frames/s*	Priority: 6, Rate: 100Mbit, Ceil: 100Mbit.
Entity_02	*30,84 frames/s*	Priority: 7, Rate: 100Kbit, Ceil: 100Kbit.
Entity_02	*66,41 frames/s*	Priority: 4, Rate: 300Kbit, Ceil: 300Kbit.

Both scenarios (a) and (b) presented in Figure 9 were evaluated concomitantly using the same time interval, and all the results were collected in real-time.

5 Concluding Remarks

This work presented the design and prototyping of SDN switch with an application-driven approach. This proposal is a Linux-based SDN switch that allows the creation and modifies of forwarding rules in the SDN data plane. This solution has a fine granularity concerning other solutions used in SDN networks, allowing adjusting the application-oriented QoS parameters, not by flow or port as most legacy switches.

The flexibility added to the Software layer makes it easy to add new features and new support to new SDN architectures. The ETSCP guides a Switch to Workspace support and changes in operation time on ETArch architecture. The NEA_SWITCHD module acts in the orchestration of the Linux Kernel modules responsible for the programmability.

The results presented in this work show the ability of the SDN Switch to receive control messages from DTSA and act in the configuration of QoS policies (priority and bandwidth) in real-time. To the road ahead, we look forward to using high-

performance hardware as the switch fabric implementation co-designed in a Kernel Linux embedded software environment.

References

1. de Oliveira Silva, F., Gonçalves, M. A., de Souza Pereira, J. H., Pasquini, R., Rosa, P. F., and Kofuji, S. T.: On the analysis of multicast traffic over the entity title architecture. In 2012 18th IEEE International Conference on Networks (ICON) (pp. 30-35). IEEE (2012)
2. de Oliveira Silva, F., de Souza Pereira, J. H., Rosa, P. F., and Kofuji, S. T.: Enabling future internet architecture research and experimentation by using software defined networking. In 2012 European Workshop on Software Defined Networking (pp. 73-78). IEEE (2012)
3. Kalyaev, A., and Melnik, E.: FPGA-based approach for organization of SDN switch. In 2015 9th International Conference on Application of Information and Communication Technologies (AICT) (pp. 363-366). IEEE (2015)
4. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., and Turner, J.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2), 69-74. ACM (2008)
5. Farhad, H., Lee, H., and Nakao, A.: Data plane programmability in SDN. In 2014 IEEE 22nd International Conference on Network Protocols (pp. 583-588). IEEE (2014).
6. Bifulco, R., and Rétvári, G.: A survey on the programmable data plane: Abstractions, architectures, and open problems. In 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR) (pp. 1-7). IEEE (2018)
7. Freitas, M. S., Rosa, P. F., and de Oliveira Silva, F.: ConForm: In-Band Control Flows Self-establishment with Integrated Topology Discovery to SDN-Based Networks. In Workshops of the International Conference on Advanced Information Networking and Applications (pp. 100-109). Springer, Cham (2020).
8. Oliveira, R. D., Molinos, D. N., Freitas, M. S., Rosa, P. F., and de Oliveira Silva, F.: Workspace-based Virtual Networks: A Clean Slate Approach to Slicing Cloud Networks. In CLOSER (pp. 464-470) (2019)
9. Bosshart, Pat and Daly, Dan and Gibb, Glen and Izzard, Martin and McKeown, Nick and Rexford, Jennifer and Schlesinger, Cole and Talayco, Dan and Vahdat, Amin and Varghese, George and Walker, David (2014), P4: Programming Protocol-independent Packet Processors, ACM SIGCOMM Computer Communication Review. New York, NY, USA, doi:10.1145/2656877.2656890
10. Mehra, M., Maurya, S., and Tiwari, N. K. (2019). Network load balancing in software defined network: A survey. International Journal of Applied Engineering Research, 14(2), 245-253
11. Oliveira, R. D., Freitas, M. S., Molinos, D. N., Rosa, P. F., and Mesquita, D. G. (2021, May). ETSCP: Flexible SDN data plane configuration based on bootstrapping of in-band control channels. In 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM) (pp. 711-715). IEEE.
12. IEEE. Ieee standard for ethernet, IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015), pp. 1–5600, Aug 2018.
13. Kim, H., and Feamster, N. (2013). Improving network management with software defined networking. IEEE Communications Magazine, 51(2), 114-119.
14. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., ... and Walker, D. (2014). P4: Programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review, 44(3), 87-95.
15. OF-CONFIG 1.2 (2014). In: OpenFlow Management and Configuration Protocol. Open Networking Foundation. Available via DIALOG. https://opennetworking.org/wp-content/uploads/2013/02/of-config-1.2.pdf