A Robust SDN Network Architecture for Service Providers

Fernando López-Rodríguez
Department of Electrical Engineering
Universidade de Brasília (UnB)
Brasília, Brazil
Email: fernando.lopez.br@ieee.org

Divanilson R. Campelo
Centro de Informática (CIn)
Universidade Federal de Pernambuco (UFPE)
Recife, Brazil
Email: dcampelo@cin.ufpe.br

Abstract—**Large scale networks, such as those deployed by Service Providers (SPs), employ robust architectures, capable of supporting large volumes of traffic with very different characteristics. Their network equipment has significant processing load, being responsible for building both a routing logic and the routing of traffic itself. By having the network control implemented in a distributed manner and being built with a limited number of vendors, these networks have limitations of control and traffic engineering, hindering the differentiation among SPs. Additionally, the network intelligence is hidden in the network equipment, making innovations very slow and conditioned to the vendors interests. As an alternative option, this work proposes a Software Defined Networking (SDN)-OpenFlow network architecture that attempts to improve the previously mentioned problems, and, at the same time, to solve the arising difficulties related to the SDN network centralized feature. With the proposed architecture, a robust SP SDN-OpenFlow network is created to support high controller response times and controller outages, without additional delays in the creation of flows and with significant reduction of the controller load. A prototype has been built using Open vSwitch as a virtualization software for OpenFlow clients, Mininet for the topology construction and Ryu as the controller, all with OpenFlow 1.3 support. The obtained results are general and can be extended to other types of networks.**

I. Introduction

Service Providers (SPs) are organizations that commercialize Internet access for enterprises, end users and other SPs. In addition to access services, SPs usually offer a diversity of other services such as cloud storage, VoIP (Voice over IP) services, digital television, among others. Due to these characteristics, SPs must carry a huge volume of traffic with very different QoS (Quality of Service) requirements [1], making their network management a complex task.

In architectural terms, SP networks have automatic, distributed control mechanisms such as routing protocols (e.g., Internal Border Gateway Protocol – IBGP, Open Shortest Path First – OSPF, Intermediate System to Intermediate System – ISIS), and signalling protocols (e.g., Label Distribution Protocol – LDP, Resource Reservation Protocol – RSVP). These mechanisms generate very robust networks, with equipment able to automatically recalculate the topology when a network change occurs. However, those mechanisms also lead to complex equipment, responsible for building routing tables and routing traffic, what increment their processing load. As additional features, SP networks have high infrastructure and operation costs, and usually must keep different management teams for each type of network, such as IP (Internet Protocol) and WDM (Wavelength Division Multiplexing). Their equipment is from a reduced group of vendors with a similar set of functionalities, what reduces the possibility of traffic engineering and the differentiation among SPs. The network “intelligence” is hidden inside the equipment, slowing innovation and conditioning it to the network manufacturers interests.

As an alternative way, Software Defined Networking (SDN), along with the OpenFlow protocol, provides an open interface through which client devices, the OpenFlow clients, interact with a network controller, the OpenFlow controller [2], [3]. The controller has a global view of the network and is responsible for managing all flow tables at each OpenFlow client. SDN enables the separation of the data plane (formed by the OpenFlow clients) from the control plane (i.e., OpenFlow controller), providing a significant number of opportunities such as traffic engineering [4], [5], [6], control plane unification for different kinds of networks such as IP and WDM [7], [8], mobility management [9], and several others. These possibilities have attracted the interest of many network equipment vendors, some of which are already incorporating OpenFlow in their products [10].

In consonance with this trend, this paper presents a modified SDN-OpenFlow network architecture for service providers, even though the obtained results are general and can be extended to other types of networks. For an SP, the standard SDN-OpenFlow architecture does not seem to be appropriate due to the following reasons: a) its centralized characteristic, which requires that each new flow has to be processed by the controller, generating overload on the controller; b) its excessive dependence on the controller, which greatly reduces the network robustness; and c) the performance degradation due to the fact that the creation of each new flow has to wait for the controller response for routing traffic. This paper proposes solutions to overcome each of these problems through a network architecture that is able to manage traffic with and without QoS. To verify the effectiveness of the proposed architecture, a prototype has been built with Open vSwitch, Mininet and the Ryu controller, all with support to OpenFlow 1.3. In addition, the paper also proposes an alternative architecture with hybrid equipment, which preserves the capabilities of the distributed control plane and at the same time incorporates the OpenFlow functionalities. The paper is summarized with a qualitative comparison in terms of some network performance metrics among the current classic network, the standard SDN-OpenFlow network, the proposed hybrid network and the proposed SDN-OpenFlow network architecture.

The rest of this paper is organized as follows. Section II presents the SDN-OpenFlow proposed architecture, highlighting the general operation logic, the proactive mechanisms that allow to reach all target networks, and the QoS management logic that makes possible the creation of new flows without additional delays. In section III, an alternative architecture with hybrid equipment is suggested. In section IV, the prototype of the proposed SDN-OpenFlow architecture and its associated results are presented, along with a comparative table among the four aforementioned network architectures in terms of some network performance metrics. Finally, Section V presents the main conclusions of the paper.

II. Proposed SDN-OpenFlow Architecture

A. General considerations

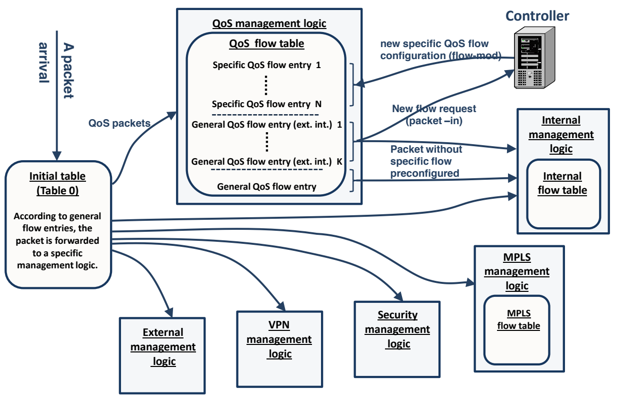
This paper proposes an SP SDN network architecture based on OpenFlow that uses the Multiprotocol Label Switching (MPLS) data plane [8]. The MPLS technology already provides an appropriate data plane abstraction, since it uses the concept of flows, which is suitable to OpenFlow. This protocol has been developed in this direction, incorporating the necessary functionalities to allow the use of the MPLS data plane [3]. When a packet is processed by a flow table in an OpenFlow client, the packet can match a flow entry according to its MPLS label, and the associated instructions can modify, add or erase MPLS labels.

The fact that OpenFlow allows the control of individual flows enhances the accuracy and the potentials of the management capabilities. However, this individual manipulation also leads to a larger processing delay and network overhead. Furthermore, the creation of each new flow requires the corresponding flow entry configuration in the flow tables of each equipment involved in the end-to-end path, with the corresponding controller-clients communications. For this reason, it is important to define policies for the different types of traffic, analysing which one requires a flow by flow treatment or a general treatment to reduce the controller overhead.

B. General operation logic

In each OpenFlow client of the proposed SDN-OpenFlow architecture, there is an initial table indexed by 0, according to Fig.1. In table 0, general rules are applied to classify arriving packets with the smallest number of flow entries (examples of matching fields: such as differentiated services code point (DSCP) field, MPLS experimentals bits (EXP), port group, IPs address group), and the packets are forwarded to a specific management logic. The specific management logics utilized in the proposed architecture are:

1) *Internal management logic*: It allows to reach all internal networks to the SP, such as interconnection networks, service networks, and both residential and enterprise customers networks. The result of this management logic is similar to the one obtained with today's internal routing protocols.



A packet arrival
QoS packets
Initial table (Table 0)
According to general flow entries, the packet is forwarded to a specific management logic.
QoS management logic
QoS flow table
Specific QoS flow entry 1
Specific QoS flow entry N
General QoS flow entry (ext. int.) 1
General QoS flow entry (ext. int.) K
General QoS flow entry
Controller
new specific QoS flow configuration (flow-mod)
New flow request (packet –in)
Packet without specific flow preconfigured
Internal management logic
Internal flow table
MPLS management logic
MPLS flow table
External management logic
VPN management logic
Security management logic

Fig. 1. General architecture and QoS management logic for border equipment.

2) *MPLS management logic*: It is the case when an incoming packet is routed based on its MPLS label. The result of this management logic is similar to the one obtained with protocols such as RSVP or LDP.
3) *QoS management logic*: It allows to manipulate individual flows that require end-to-end QoS. Examples are VoIP, video, digital TV and real time control applications. This management logic is one of the main contributions of this paper.
4) Other management logics not developed in this paper as: External routing, which allows to reach all target networks outside the autonomous system; Virtual Private Network (VPN) logic, which allows customers to interconnect their sites maintaining isolation policies; security logic, used for traffic that needs to be authorized before being sent through the network.

Each management logic is implemented by a flow table or a group of linked flow tables, which are located in the OpenFlow pipeline processing of each equipment. These tables differ according to the kind of OpenFlow client and its location in the network. Next subsection describes the internal and MPLS management logics in a general way, and the QoS management logic, the focus of this paper, with more details.

C. Internal and MPLS management logic

These logics are composed by tables with forwarding information to reach all internal networks to the SP, similar to tables of existing distributed control networks (i.e., without a flow-by-flow basis). These tables are proactively built by the controller (i.e., before the arrival of flows) at each OpenFlow client, and are updated by the controller when topological changes occur in the network (the OpenFlow client must notify the controller about the changes). The controller must have the information of all directly connected networks to each OpenFlow client and the network topology.

With the topological knowledge and an appropriate routing protocol, the controller is able to the execute the internal routing protocol (e.g., OSPF, or ISIS, or a new one), find out how to reach all internal networks from each OpenFlow client and build the internal flow tables in each OpenFlow client, constructing, in this way, the internal management logic. After that, the controller establishes the association between the MPLS labels and the destination networks, producing the Forwarding Equivalent Classes (FECs). The MPLS tables are built in each OpenFlow and then the MPLS management logic. (similar to what occurs in current distributed control networks with LDP and RSVP protocols). For the internal and MPLS management logics, in the cases where the routing protocols executed at the controller find several optimal paths, they can be considered and added in the corresponding flow tables. For this purpose, OpenFlow allows to configure *Group Tables* that are able to route traffic with a suitable balancing algorithm and improve the network load distribution [3].

Three important considerations must be highlighted. The first one is that, like in all SDN architectures, the routing algorithm is only known and implemented in the controller. The controller is responsible for building all routing tables for each node. Second, the OpenFlow clients do not need to exchange information about routing and topological changes among themselves, reducing the convergence time; all OpenFlow clients only need to inform topological changes to the controller and the latter recalculates the routing table, constructing a new topology. Last, it is not necessary to use complex routing and signalling protocols such as OSPF Traffic Engineering (OSPF-TE) , ISIS Traffic Engineering (ISIS-TE), LDP or RSVP.

D. QoS management logic

When a packet with QoS requirements enters the network, the QoS management logic reactively creates an end-to-end path that allows to satisfy such QoS requirements of the specific flow. To identify when a packet enters the network, the border equipment (the ones that have some interfaces inside and others outside the SDN-OpenFlow network) and the internal equipment (those ones that have all interfaces inside the SDN-OpenFlow network) must be identified. To understand how this logic works, the two types of flow entries of the QoS tables are firstly described:

- General flow entries: Allow any type of QoS traffic to find matching using few flow entries. In the border equipment, the general flow entries treat differently if the packet is entering or exiting from the OpenFlow network (it is done by the examination of the packet input interface). These flow entries allow to forward the first packets of each QoS flow, and they are used while the packet does not have a specific QoS flow entry yet.
- Specific flow entries: These ones are specific to each individual QoS flow. Initially, the QoS table does not have specific flow entries; they will be created during the operation.

When a new QoS packet enters the network through an border equipment, the packet is sent to the QoS table and is initially processed by a general QoS entry with external input interfaces. This general flow forwards the packet through two different processing pipelines (see Figs. 1 and 2) simultaneously, as described below:

1 – Firstly, the general flow entry with external input interfaces sends a complete or partial copy of the packet to the controller (a *packet-in* message). To do it, the OpenFlow client has the immediate execution *instruction* of the type *Apply-Actions*, and within it, an *action* of type *Output* to the reserved port *Controller* [3]. With this information, the controller calculates the optimal path to satisfy the QoS flow requirements. One can use a constrained Dijkstra algorithm similar to those used by ISIS-TE or OSPF-TE, or create a new one specifically designed for each QoS type [4]. The controller configures a specific new flow entry through a *flow-mod* message in the QoS table of each equipment of the optimal path (built in the opposite direction of flow traffic). It is important to point out that all the specific QoS flow entries are created with high priority (to be matched before general entries) and with an appropriate idle time-out (to be automatically deleted after a period of inactivity). Additionally, since OpenFlow has priority queues, the QoS flows can be mapped to these queues. Finally, the controller sends a *packet-out* message without action, indicating that the packet that originated the *packet-in* message must be dropped (this is because the packet has already been sent by the second pipeline processing, described below).

2 – Concurrently with the actions described in the previous paragraph, the packet continues the pipeline processing indicated in the general QoS flow entry with external input interface. In this flow entry, an *instruction* of *Goto-Table* type is defined [3], which instructs that the packet must continue its processing in the internal flow table (see Figs. 1 and 2). Thereby, the packet is processed immediately as a flow without QoS and it does not have to wait for the controller response to be forwarded.

When the packet enters an internal equipment or an border equipment (output border equipment) through an internal input interface, the general QoS flow entries are simpler, since it only has to send the packet to the internal flow table (second pipeline processing). This is because the input border equipment has already sent a request for a new flow to the controller, and the controller creates a specific QoS flow entry in all involved equipment, including the internal equipment and the output border equipment (avoiding a redundant new flow request).

E. Important remarks about the proposed architecture.

It is important to highlight that with the QoS management logic, the controller does not have to add waiting time in the establishment of a new flow. This statement is based on the fact that packets from the same flow are always initially forwarded by the internal flow table of the internal management logic, and are simultaneously processed by the controller to find the optimal path. Afterwards, when the QoS table has the specific QoS flow, all subsequent packets of this flow are matched with this new specific flow entry and are forwarded by the optimal specific QoS path. Load balancing is not allowed in the specific QoS path. Therefore, a unique path for each QoS flow is obtained, reducing the QoS traffic jitter.

In addition, the proposed SDN-OpenFlow architecture allows to reduce the excessive controller dependence that the standard SDN-OpenFlow implementations have. When the controller is in outage or presents high response times, the new QoS flow always continues to be forwarded trough the internal management logic. This is a very important aspect of robustness, which is necessary for all SPs.

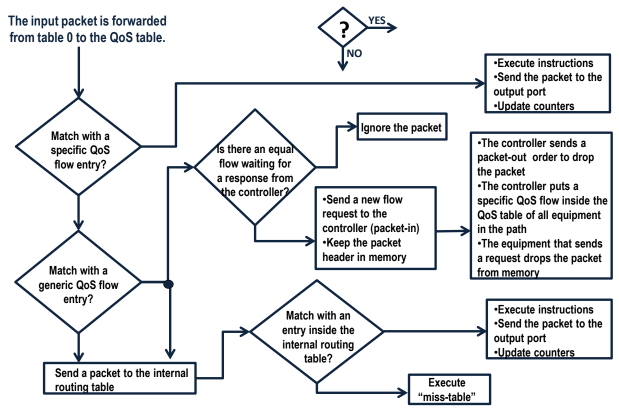

The input packet is forwarded from table 0 to the QoS table.
?
YES
NO
Match with a specific QoS flow entry?
•Execute instructions
•Send the packet to the output port
•Update counters
Is there an equal flow waiting for a response from the controller?
Ignore the packet
•Send a new flow request to the controller (packet-in)
•Keep the packet header in memory
•The controller sends a packet-out order to drop the packet
•The controller puts a specific QoS flow inside the QoS table of all equipment in the path
•The equipment that sends a request drops the packet from memory
Match with a generic QoS flow entry?
Match with an entry inside the internal routing table?
•Execute instructions
•Send the packet to the output port
•Update counters
Send a packet to the internal routing table
Execute "miss-table"

Fig. 2. QoS management logic for border equipment.

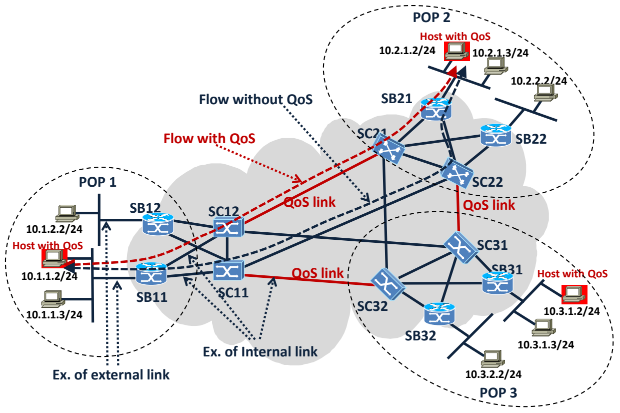


Fig. 3. Prototype topology.

Moreover, for any SDN-OpenFlow implementation, it is useful to have mechanisms for OpenFlow clients to prevent that successive packets of the same flow generate the same queries to the controller while the client is waiting for the first controller response. In the standard SDN-OpenFlow architecture, it is difficult to implement this mechanism, because all successive packets from the same flow that are waiting for the controller response are required to be stored in memory until the *packet-out* message arrives and the output action of the *packet-out* message must be applied to all stored packets. In the proposed SDN-OpenFlow architecture, only the header of the first packet for each QoS flow must be stored, because the successive packets of the same flow have already been sent when the *packet-out* arrives. In the proposed architecture, only the header of entering packet must be compared with the headers of the storage packets, and a *packet-in* message is generated if there is not a match (first packet of a new flow, Fig. 2).

According to [11], the probability of finding a packet belonging to a new flow in a OpenFlow client is 4%, which means that with the standard OpenFlow implementation 4% of the total traffic is forwarded to the controller, which is very excessive for a SP. In the proposed SDN-OpenFlow architecture, the controller load is significantly reduced because: 1) the traffic without QoS is not forwarded to the controller; 2) only the first packet of each QoS flow generates a *packet-in* message; and 3) only the input border equipment sends a new QoS flow request to the controller.

III. Proposed Architecture with Hybrid Equipment

In this Section, an alternative architecture that uses hybrid equipment is considered. Such an architecture maintains the capabilities of the distributed control plane and at the same time incorporates the OpenFlow capabilities. For this, OpenFlow has the instruction *Write-actions* that allows to define in the *action-set* the action *output* to the reserved port *Normal* [3]. This action allows the OpenFlow pipeline to forward the packet to the classic distributed control plane.

Combining hybrid equipment with the SDN-OpenFlow architecture presented in Section II, it is possible to develop an intermediate architecture that is able to program table 0 and the QoS management logic with OpenFlow, as well as the other management logics with the distributed control plane. In this case, the operation is as follows: If a packet requiring QoS arrives to table 0, it is sent to the QoS management logic, as indicated in section II. If the QoS table does not have a specific flow entry yet, the packet is forwarded to the controller and simultaneously is sent to internal management logic, which is implemented in this case by the distributed control plane (using the reserved port *Normal*). While there is not a specific QoS flow entry in the QoS flow table, successive packets of the same flow are forwarded by the distributed control plane. When the specific QoS flow entry is available, the packet is forwarded by this specific QoS flow entry.

This architecture with hybrid equipment would have little resistance to be used in a SP. This architecture increases the possibilities for traffic engineering, and at the same time, completely maintains the network robustness, since when there is a controller outage, the traffic can be indefinitely routed with the classic distributed control plane. This architecture could be used in a migration process from distributed control plane to the SDN-OpenFlow architecture proposed in the paper.

IV. Prototype

To demonstrate the benefits provided by the SDN-OpenFlow proposed architecture, a prototype with the topology shown in Fig. 3 has been created. It consists of 3 points of presence (POPs), each one with two internal equipment (SCxx), two border equipment (SBxx) and three hosts. The prototype has been created with Mininet 2.1 for the topology construction, Open vSwitch 2.0 as a software switch that implements the OpenFlow clients, and Ryu 3.3 as the controller, all with support to OpenFlow 1.3. Without loss of generality, the topology has been implemented without MPLS, with the purpose of facilitating the implementation of the logics and reducing the number of tables used.

The QoS links shown in Fig. 3 are only used for QoS traffic, which is generated by the hosts with IP address 10.x.1.2 (x ranging from 1 to 3) and it has the DSCP field set as 5. The QoS flow table is named as table 5, and the internal flow table is named as table 10 (table 5 redirects to table 10, but the opposite is not possible according to OpenFlow).

A. General flow entry construction

The initially, proactively created flows by the controller are described below.

Table 0:

- In SBx1, the layer 2 flow entries are inserted to make possible the interconnection among hosts inside the same Local Area Network (LAN).
- In each equipment, two flow entries are added: one to send the packets marked with DSCP = 0 to table 10 (internal flow table), and another to send the packets marked with DSCP = 5 to table 5 (QoS flow table).

Table 10:

- It contains a group of flow entries to forward the packets to all target networks (similar to current routing protocols).

Table 5:

- In the border equipment, the general flow entries with external input interfaces (input border equipment) are inserted. These general entries send the packet to two pipelines: to the controller (instruction *apply-action*, with reserved port *Controller*); and to table 10 (instruction *Goto-table 10*) for immediate processing.
- In the internal equipment and in the border equipment, the default QoS general flow entries are inserted, which only send the packet to table 10 (the internal and the output border equipment do not send a new flow request to the controller to avoid request duplication).

B. Operation mode and specific flow entry

Initially, all flows are processed by table 0, which sends the QoS flow (DSCP = 5) to table 5 (Fig. 4, option – OP. 1), and sends the flows without QoS requirements to table 10 (Fig. 4, OP. 2).

In the SBxx equipment, when there is not a specific QoS flow entry yet, and there is a match with a general QoS flow entry with external input interface (Fig.4, OP. 1.b), the SBxx sends a *packet-in* message to the controller and redirects the packet to table 10. The controller receives the message and responds with a *flow-mod* message, creating a specific QoS flow (inside all tables 5 of every equipment involved in the QoS path) with idle-timeout of 60 seconds and with priority higher than that of the general QoS flow entries. Additionally, the controller sends a *packet-out* message to the SBxx equipment requesting it to drop the packet (its packet has just been processed by table 10).

In the SCxx, or SBxx equipment, when there is not a specific QoS flow yet, and there is not a match with a general QoS flow entry with external input interface (Fig. 4, OP. 1.c), it only sends the packet to table 10. Finally, in all equipment, when there exists an specific QoS flow entry (Fig. 4, OP. 1.a) this one processes the packet.

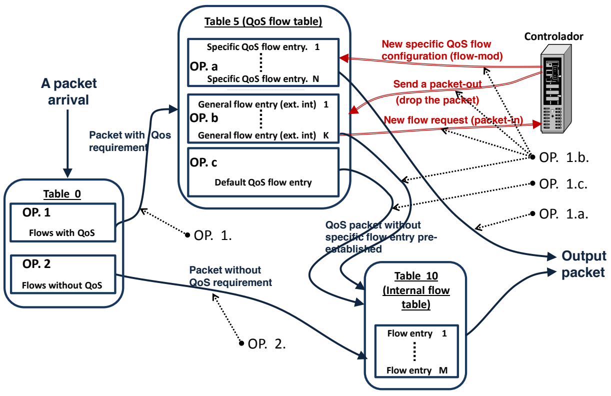


Table 5 (QoS flow table)
Specific QoS flow entry. 1
OP. a
Specific QoS flow entry. N
General flow entry (ext. int) 1
OP. b
General flow entry (ext. int) K
OP. c
Default QoS flow entry
Controlador
New specific QoS flow configuration (flow-mod)
Send a packet-out (drop the packet)
New flow request (packet-in)
A packet arrival
Packet with Qos requirement
OP. 1.b.
OP. 1.c.
OP. 1.a.
Table 0
OP. 1
Flows with QoS
OP. 2
Flows without QoS
OP. 1.
QoS packet without specific flow entry pre-established
Output packet
Packet without QoS requirement
Table 10 (Internal flow table)
Flow entry 1
Flow entry M
OP. 2.

Fig. 4. Prototype architecture for a SBxx equipment.

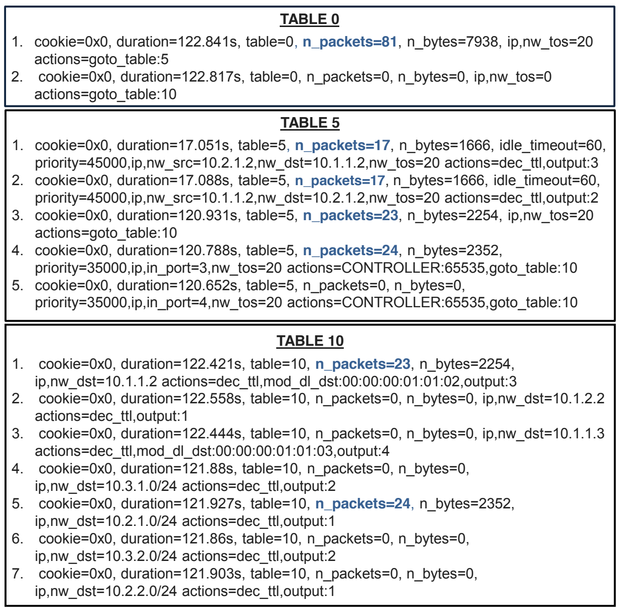
C. Tests

First, it is verified that all destination networks are reachable using table 10. Then the QoS flow generation between the hosts with IP addresses 10.1.1.2 and 10.2.1.2 is tested, verifying that only the first round trip packet is processed by table 10 and the remaining packets of the same flow are processed by the specific QoS flow entry (reactively constructed).

After that, a controller outage is emulated. During the outage, a QoS flow between hosts with IP 10.1.1.2 and 10.2.1.2 is generated (flow without QoS in Fig. 3). Finally, the controller is re-established and a specific QoS flow is created (flow with QoS in Fig. 3). Fig. 5 shows the SB11 real flow tables regrouped, obtained with the command *sudo ovs-ofctl O OpenFlow13 dump-ow sb11*. In this example, 81 QoS packets (all with DSCP = 5) are generated (41 in SB11-SB21 direction, and 40 in SB21-SB11 direction), matching with flow entry 1 of table 0, which sends the packet to table 5. From the 81 packets, 45 are sent during the controller outage (23 in SB11-SB21 direction, and 22 in SB21-SB11 direction), and the other 36 (18 in each direction) are sent with the controller re-established.

It is important to notice that packet 24 in the SB11-SB21 direction, and packet 23 in the SB21-SB11 direction are the first packets that have a controller response, with the creation of two specific QoS flow entries, one in each direction. However, these packets are still processed by the general QoS flow entry with external input interface. As a result, the first 24 packets in SB11-SB21 direction are processed by the general QoS flow entry with external input interface (Fig. 5, table 5, flow entry 4). This general QoS flow entry sends the packets simultaneously to the controller and to table 10, where they match with flow entry 5 of table 10. On the other hand, the 23 first packets in SB21-SB11 direction are processed by the general QoS flow entry without external input interface (Fig. 5, table 5, flow entry 3), because SB11 is an output border router in SB21-SB11 direction. The flow entry must send the packets only to table 10, where they match with flow entry 1. The last 17 packets in both directions have already an specific QoS flow entry in QoS table (Fig. 5, table 5, flow entries 1 and 2), then they are directly routed for these specific ow entries.

The tests confirm the robustness of the proposed SDN-



```
TABLE 0
1. cookie=0x0, duration=122.841s, table=0, n_packets=81, n_bytes=7938, ip,nw_tos=20
   actions=goto_table:5
2.  cookie=0x0, duration=122.817s, table=0, n_packets=0, n_bytes=0, ip,nw_tos=0
   actions=goto_table:10

TABLE 5
1. cookie=0x0, duration=17.051s, table=5, n_packets=17, n_bytes=1666, idle_timeout=60,
   priority=45000,ip,nw_src=10.2.1.2,nw_dst=10.1.1.2,nw_tos=20 actions=dec_ttl,output:3
2. cookie=0x0, duration=17.088s, table=5, n_packets=17, n_bytes=1666, idle_timeout=60,
   priority=45000,ip,nw_src=10.1.1.2,nw_dst=10.2.1.2,nw_tos=20 actions=dec_ttl,output:2
3. cookie=0x0, duration=120.931s, table=5, n_packets=23, n_bytes=2254, ip,nw_tos=20
   actions=goto_table:10
4. cookie=0x0, duration=120.788s, table=5, n_packets=24, n_bytes=2352,
   priority=35000,ip,in_port=3,nw_tos=20 actions=CONTROLLER:65535,goto_table:10
5. cookie=0x0, duration=120.652s, table=5, n_packets=0, n_bytes=0,
   priority=35000,ip,in_port=4,nw_tos=20 actions=CONTROLLER:65535,goto_table:10

TABLE 10
1.  cookie=0x0, duration=122.421s, table=10, n_packets=23, n_bytes=2254,
   ip,nw_dst=10.1.1.2 actions=dec_ttl,mod_dl_dst:00:00:00:01:01:02,output:3
2.  cookie=0x0, duration=122.558s, table=10, n_packets=0, n_bytes=0, ip,nw_dst=10.1.2.2
   actions=dec_ttl,output:1
3.  cookie=0x0, duration=122.444s, table=10, n_packets=0, n_bytes=0, ip,nw_dst=10.1.1.3
   actions=dec_ttl,mod_dl_dst:00:00:00:01:01:03,output:4
4.  cookie=0x0, duration=121.88s, table=10, n_packets=0, n_bytes=0,
   ip,nw_dst=10.3.1.0/24 actions=dec_ttl,output:2
5.  cookie=0x0, duration=121.927s, table=10, n_packets=24, n_bytes=2352,
   ip,nw_dst=10.2.1.0/24 actions=dec_ttl,output:1
6.  cookie=0x0, duration=121.86s, table=10, n_packets=0, n_bytes=0,
   ip,nw_dst=10.3.2.0/24 actions=dec_ttl,output:2
7.  cookie=0x0, duration=121.903s, table=10, n_packets=0, n_bytes=0,
   ip,nw_dst=10.2.2.0/24 actions=dec_ttl,output:1
```

Fig. 5. Regrouped real flow tables (for the SB11 equipment)

OpenFlow architecture, and show that the packet continues to be routed even if there is a controller outage. Additionally, when the controller is re-established, the specific QoS flow entry is created, and the flow is re-routed. Finally, it is confirmed the immediate routing of all packets and that the controller does not add waiting times in the creation of a new flow. The proposed SDN-OpenFlow architecture leads to important improvements over the standard SDN-OpenFlow architecture, and at the same time maintains the potentials that OpenFlow provides.

TABLE I. COMPARISON OF NETWORK ARCHITECTURES FOR SERVICE PROVIDERS

	Current distributed control network	Standard SDN-OpenFlow network	Hybrid network proposal	SDN-OpenFlow network proposal
Forwarding response	Immediate	Interme-diate	Immediate	Immediate
Traffic engineering possibilities	Good	Very good	Very good	Very good
Jitter for QoS traffic	Interme-diate	Low	Low	Low
Network behavior during controller outage	N/A	Poor	Excellent	Very Good
Network behavior with high response times of the controller	N/A	Poor	Excellent	Very Good
Controller overload	N/A	Poor	Excellent	Very good
Routing protocols complexity	Very high	Interme-diate	Very high	Interme-diate
Processing load in network equipment	High	Low	High	Low

To summarize the discussions and results presented in this paper, Table I shows a qualitative comparison among the current distributed control networks, the standard SDN-OpenFlow networks, the proposed hybrid network, and the proposed SDN-OpenFlow architecture in terms of some important metrics for service providers. It can be noted that the proposed hybrid architecture and the proposed SDN-OpenFlow architecture have the best characteristics, and, therefore, are promising alternatives to current distributed control networks and standard SDN-OpenFlow networks.

V. CONCLUSION

The proposed SDN-OpenFlow architecture improves critical aspects of the standard OpenFlow architecture, such as the excessive dependence on the controller, the additional delay added by the controller in the creation of a new flow, and the excessive information that must be processed by the controller. At the same time, it allows to solve today's networks problems, such as the impossibility to build massively individual paths for QoS flows, traffic engineering limitations, complex information distribution algorithms, significant jitter, the impossibility to unify the control plan to different networks, and slow innovation conditioned to the vendor interest. An alternative hybrid architecture that preserves the capabilities of the distributed control and incorporates the OpenFlow capabilities has been presented.

ACKNOWLEDGMENT

This work has been supported by CNPq and CAPES.

REFERENCES

[1] J. L. García-Dorado, A. Finamore, M. Mellia, M. Meo, and M. Munafo, "Characterization of isp traffic: Trends, user habits, and access technology impact," *Network and Service Management, IEEE Transactions on*, vol. 9, no. 2, pp. 142–155, 2012.

[2] 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, pp. 69–74, 2008.

[3] Open Networking Foundation. (2002) Openflow switch specification version.1.3.0. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf

[4] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over openflow networks: An optimization framework for qos routing," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*. IEEE, 2011, pp. 2241–2244.

[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 *National Fiber Optic Engineers Conference*. Optical Society of America, 2011, p. NThD3.

[6] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 2211–2219.

[7] S. Das, "Pac.c," PhD Dissertation, Stanford University, 2012, ⟨http://archive.openflow.org/wk/index.php/PACC_Thesis⟩.

[8] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and circuit network convergence with openflow," in *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*. IEEE, 2010, pp. 1–3.

[9] S. Namal, I. Ahmad, A. Gurtov, and M. Ylianttila, "Enabling secure mobility with openflow," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. IEEE, 2013, pp. 1–5.

[10] Open Networking Foundation. Sdn product directory. [Online]. Available: https://www.opennetworking.org/sdn-resources/onf-products-listing

[11] F. Wamser, R. Pries, D. Staehle, K. Heck, and P. Tran-Gia, "Traffic characterization of a residential wireless internet access," *Telecommunication Systems*, vol. 48, no. 1-2, pp. 5–17, 2011.