Engineering and Technology Journal e-ISSN: 2456-3358
Volume 08 Issue 08 August -2023, Page No.- 2540-2555
DOI: 10.47191/etj/v8i8.08, I.F. – 7.136

Exploring Routing and Quality of Service in Software-Defined Networks: Interactions with Legacy Systems and Flow Management

Daniel Alberto Priano[1], Fabio Sergio Bruschetti[2], María Claudia Abeledo[3], Javier Guevara[4]
[1,2,3,4] Universidad Nacional de San Martín

ABSTRACT: The following article presents the most important aspects of routing technologies and the QoS (Quality of Service) of the SDN architecture. The interaction of SDN with traditional or legacy networks is discussed in depth and concepts related to flow management are developed. Additionally, the analysis of the Open Flow protocol is included.

KEYWORDS: SDN, OpenFlow, quality of service, QoS, routing, SDN architecture, internetworking, flow-based routing

1. INTRODUCTION

For organizations, the use of software-defined networks (SDN) enables and accelerates the implementation and deployment of applications while significantly reducing IT (Information Technology) costs by automating workflows established by the policies applicable to data networks. SDN technology enables the development of cloud architectures by automating on-demand automated delivery of applications or mobility at scale. SDN enhances the benefits of data center virtualization by increasing the flexibility and resource utilization and reducing costs and overhead.

SDN achieves these goals by converging the management of network services and applications into an extensible and centralized deployment platforms that can automate the provisioning and configuration of the entire infrastructure. TE (Traffic Engineering) policies can be applied to different workflows. The result is a modern and agile infrastructure that can quickly deliver new applications and services instead of days or weeks required in the past.

This new approach to software-defined networks proposes five fundamental aspects:

- Separation of planes (control, data and management),
- Programmable and low-cost devices,
- Centralized control (through a controller that can manage such devices),
- Network virtualization and automation,
- Open-source code and structures.

On the other hand, when experimenting with large-scale topologies in SDN networks, we can observe many advantages in routing recovery. Compared to SDN routing, routing convergence in the legacy network is much more influenced by link delays. When the delay of a network link is high and the network is large, SDN networks have a shorter routing convergence time than legacy networks.

Centralized control in an SDN network allows improving the efficiency of routing computation and developing fine-grained control of network packets. Some literature studies the convergence time of legacy routing mechanisms and SDN routing by measuring their performance in terms of delay, packet forwarding or convergence after link/node failure.

In this paper, the exploratory analysis of the existing literature has focused on the sources for each of the categories and subcategories of this topic developed in Europe and the US. This work has been limited to information from the last decade, as most of the research and applications based on SDN have been developed in this period.

We have consulted different sources of documentation such as IEEE Xplore[1], Google Scholar, ONF (Open Network Foundation) and ITU (International Telecommunication Union), among others. For the search, we used keywords such as SDN architecture, SDN controllers, NFV (Network Function Virtualization) for SDN, SDN applications and SDN standards.

1.1 Traffic Engineering

TE is a very important mechanism for optimizing the performance of data networks. It allows regulating the flow of data transmitted over the network based on a dynamic and predictive analysis of the behavior of the transmitted data[2]. A wide variety of TE techniques exist and most of them are implemented in MPLS (Multiprotocol Label Switching) networks.

SDN networks allow traffic engineering to be implemented in a natural way, enabling policy coordination between different ISPs by combining device performance with the big picture view[3].

At this point, we would like to point out that the implementation of TE in MPLS will be complex because it is a multiprotocol that uses a lot of network resources. In order to implement TE, it is necessary to modify the routing protocols so that they can transport the typical attributes of these

mechanisms. For example, one of them is the available bandwidth, which has to be dynamically calculated at the nodes in order to know the volume of traffic managed at each link. This data will populate the TE table needed for tunnel selection. This table will be used to manage, for example, priorities and resource reservations. We will not go in detail into this topic because it is not the objective of this work.

1.2 Quality of Service

With the emergence of multimedia applications, traffic requirements for networks have become a critical feature: low delay, low jitter or reduced packet loss.

QoS models are used to ensure that networks can transmit data sent by critical applications. They allow users to avail themselves of multimedia services of reasonable quality under a specific contract that includes the operating parameters agreed between the provider and the customer and is called SLA (Service Level Agreement)[4].

In general, implementing QoS in routers of an IP network consists of the following phases:

- Packets markup with a code obtained from the headers of the current packets. It will be used to group packages with some criteria or rule.
- Differentiated treatment of marked packets for traffic monitoring and SLA compliance. This is achieved through bandwidth measurement, packet remarking, queuing and buffering treatment.

Two of the architectures used are: Integrated Services (IntServ) and Differentiated Services (DiffServ).

The objectives of IntServ are to preserve the datagram model of IP-based networks and to support resource reservation for real-time applications. Routers have to store certain state information of each flow and resource reservations with the RSVP (Resource Reservation Protocol)[5] in all nodes of the path. Packets of the same flow will use the same path, emulating circuit switching.

To apply QoS to a flow, it must pass through an admission check that verifies the availability of resources at each router on its path. If the resources are available, the flow is admitted to the network and each router stores its state information (soft state). This will ensure that the flow reaches its destination. To maintain these resource reservations during the transmission of flows, resource reservation requests must be renewed periodically generating an increase in network traffic.

Since the flows share the available network resources, we can say that the links will be shared between different flows. For example, the network bandwidth is shared by several flows that may contain different types of traffic.

On the other hand, the main idea of DiffServ is to classify the different flows into classes and then apply QoS rules to them. Each node operating within a DiffServ domain will follow the same common service provisioning policies and the same set of PHB (Per Hop Behavior) groups[6].

A flow normally enters a DiffServ domain through a border node. The border node performs several tasks such classification, marking, policy enforcement and traffic adaptation. These tasks are performed according to the TCA (Traffic Conditioning Agreement)[7]. This TCA specifies the traffic classification and profiling rules (measurement, marking, discarding and shaping). Border nodes interconnect different domains, each with its internal nodes, and implement functions to translate the different PHBs between TCA-compliant domains.

Forwarding traffic at each node meets a certain PHB. This is the differentiated treatment that each individual packet receives according to specific queuing service disciplines, whose mechanisms are not subject to standardization. PHBs are applied at each network node regardless of how end-to-end or cross-domain services are constructed, providing a particular treatment for each class of traffic.

Besides, flow admission is performed at the border nodes and can be configured manually or dynamically. In case of dynamic configuration, RSVP[8] will be used. This protocol reduces the scalability possibilities.

Perhaps the biggest difference between the two systems arises in terms of scalability over large networks. While the IntServ model allows for greater granularity in resource reservation on a per-flow basis, it uses greater bandwidth resources of the network links than the DiffServ model.

Diffserv networks classify packets according to a small number of flow groups by identifying them with the IPv4 DSCP (Differentiated Services Code Point)[9], IPv4 TOS (Type of Service) or IPv6 Traffic Class of each IP header in the packet. Therefore, in addition to eliminating the state dependency of each flow, DiffServ can implement QoS provisioning without the need for end-to-end signaling.

We can formally define QoS as a set of standards and mechanisms aimed to guarantee certain level of quality for the services used by critical applications. Using these mechanisms, network administrators can efficiently use existing resources and thus guarantee the required level of service. This avoids over-provisioning resources and over-dimensioning networks.

The concept of QoS is critical for applications and users to meets their requirements; this means that some traffic flows will need preferential treatment. The objective of QoS is to provide a service that ensures sufficient bandwidth, latency control and reduction of data loss[10].

In SDNs it is possible to centralize QoS management, leaving the routers to take care of their primary role and implementing QoS directly on the switches through the network controller. This solves the problems caused in large networks and facilitates effective QoS management for the entire network. This feature allows QoS policies to be changed dynamically, making SDNs truly useful.

1.3 Load Balancing

Today, our networks have to handle a large amount of traffic, serve thousands of customers and comply with the requirements and restrictions imposed. This is a very large workload and difficult for a single server to handle. The solution is to use multiple servers with a load balancer acting as an interface.

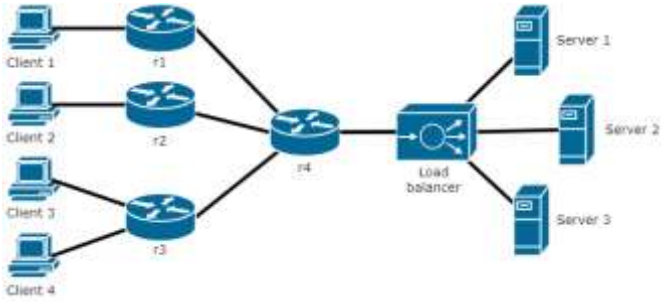

Figure 1. Load Balancing - Source: Cisco System.

As shown in Figure 1, clients send their requests to the load balancer, which receives them and forwards them to the different servers according to the defined distribution or balancing strategy.

Round Robin is one of the most common strategies. It consists of distributing requests among servers sequentially. Another strategy is Least Connections, where a new request is sent to the server with the fewest client connections at that time. There is also the IP hash strategy in which the client's IP address is used to determine the server that will serve it.

The load balancer uses dedicated hardware is expensive and is limited to what the vendor allows to be parameterized. Currently, network administrators cannot use their own load balancing algorithms since load balancers are not programmable as they are vendor locked.

The growing demand for Internet services generates sudden increases in network traffic, causing congestion and overloading of links. For example, in 2009, in a data processing center network with 150 switches and 1500 servers, the 15% of the network was congested for more than 100 seconds even though many of the links were underutilized; this implies a considerable need to optimize the allocation of network resources[11].

Most data processing centers have decided to implement a load balancer to meet the large number of users and the requirements they generate. Balancing strategies for web services (HTTP) are often based on link metrics provided during installation and do not take into account dynamic network conditions such as bandwidth utilization, packet loss and delays. There is a lack of flexibility to adjust a strategy based on different network requirements, network traffic or running applications[12].

Networks must handle large traffic volumes guaranteeing the availability of their services and avoiding unnecessary waiting times. The article "Dynamic load balancing application for servers, based on software-defined networks"[13] proposes a load balancing algorithm based on a combined criteria (available bandwidth and delay) using SDN technologies. The objective is to obtain and evaluate different network parameters at runtime. This set of parameters makes it possible to select the most responsive server among the set of servers storing and distributing the same application or providing the same service. In this way, server response time is improved by up to 50% compared to, for example, the traditional Round Robin method. SDN-based server load balancing can effectively improve server performance with low implementation complexity compared to the traditional load balancing method.

Using OpenFlow as the controller interface protocol in SDN, the load balancing of a set of servers will be based on the dynamic creation of flow tables. Through an algorithm that allows the design of dynamic flow tables, "individual flow tables" can be combined with a "group flow table". Individual tables can monitor the traffic of each client while a group table allows to classify hosts efficiently. This is also discussed in the article mentioned above.

The algorithm mentioned in the article avoids an excessive number of flow tables and also solves the defect that is generated when the number of matches in the flow table is too large. This demonstrates that it is possible to obtain better performance in network traffic scheduling.

So far, it is possible to solve some of the problems of traditional networks using SDN. SDN load balancer has certain advantages compared to the method used in traditional networks. SDN can effectively improve the performance of the load balancer and reduce the complexity of its implementation. SDN load balancers are programmable and allow you to design and implement your own or custom load balancing strategy or algorithm. Other virtues of SDN load balancers are that they do not require dedicated hardware, which saves network costs. A single switch can become a powerful load balancer through the use of SDN controllers.

1.4 Multipath routing in SDN

One of the first concepts analyzed in this section is that of applications that relate the load balancer to multipath routing; a topic about which much has been written. This routing is the most commonly implemented in SDN which will not be included in this paper, but rather an overview of its implementation and operation will be presented.

Multipath routing is a technique that uses network resources to propagate traffic from a source node to a destination node over multiple network paths. This technique is used to increase bandwidth, minimize end-to-end delay, increase fault tolerance, improve reliability, implement load balancing, among others.

There are three fundamental elements in multipath routing[14]: Path discovery, traffic distribution and route maintenance.

The emergence of SDNs helps to resolve the entropy generated by the combination of internal and external routing protocols, traffic engineering, load balancing and multipath routing in traditional networks. Thus, the idea of routing by destination to find the best route is abandoned and replaced by the search for a balanced distribution of flows. In this way, network resources and available links will be better utilized.

Efforts to achieve all this are evident in research on traffic engineering in MPLS, whose implementation and maintenance are excessively complex if dynamic network behavior is required. We could say that any solution for a new paradigm would have to be initially simple, i.e., capable of emulating the usual behavior of traditional networks in a simple way and from there evolve towards intelligent networks.

Enterprises and service providers are surrounded by a number of competing forces. The enormous growth of multimedia content, the explosion of cloud computing, the impact of increased use of mobile devices, and continuing business pressures to reduce costs are converging to wreak havoc on traditional business models as, meanwhile, revenues remain flat.

To keep pace, many companies are turning to SDN technology, revolutionizing network design and operation. SDN enables consistent management of an entire network that may contain components of complex technologies, as we will see below.

Figure 2 shows four critical areas where SDN technology can make a difference for an organization:

1. *Network programmability*: SDN allows network behavior to be controlled by software that resides beyond the network devices that provide its physical connectivity. As a result, network administrators can adapt the behavior of their networks to support new services or customers. By decoupling hardware from software, administrators can quickly introduce innovative and differentiated new services without the limitations that exist in closed, proprietary platforms.

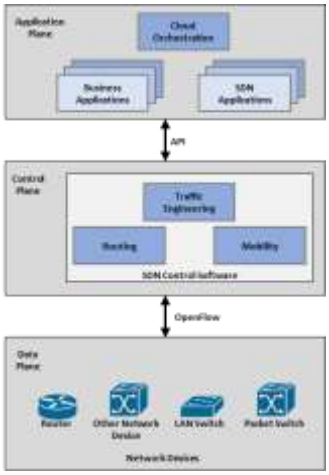

Figure 2. Logical structure of an SDN network[15].

2. *Centralization and control*: SDN are based on logically centralized network topologies that enable intelligent control and management of network resources. Devices operate autonomously with limited knowledge of the network state. With the centralized control of an SDN, a holistic view of the network is obtained where policies for bandwidth management, failover, quality of service, etc. can be optimally and intelligently implemented.
3. *Network abstraction*: services and applications running on SDN reside on the technologies and hardware that provide physical connectivity and control of the network. Applications interact with the network through APIs (Application Programming Interfaces) instead of management interfaces tightly coupled to the hardware.
4. *Openness*: SDN architectures usher in a new era of openness by enabling multi-vendor interoperability and fostering a vendor-neutral ecosystem. Openness comes from the SDN approach itself. Open APIs support a wide range of applications including cloud orchestration, OSS/BSS, SaaS, among others. These concepts are defined below:
   - Operations Support Systems (OSS) refer mainly to the network systems that are linked to its operation, for example, the configuration of its components, early detection of faults, maintenance, among others. Basically, it is what allows telecommunications network administrators to keep the service running.
   - The BSS (Business Support System) is complementary to the OSS. It allows the management of business elements through various tools for customer service, collections, invoicing, etc.[16]

- SaaS (Software as a Service) allows users to access cloud-based applications through Internet[17]. It offers an end-to-end software solution provided by a cloud service provider based on a pay-per-use model. All infrastructure, middleware, software and application data are located in the provider's data processing center. The service provider manages the hardware and software and, with the appropriate service contract, will also ensure the availability and security of applications and data.

To this must be added the critical network applications of each company or organization, which, of course, will depend on their environment. Software such as OpenFlow can provide intelligent control of multi-vendor hardware through open programmatic interfaces.

To implement the SDN concept in practice, two requirements must be met. First, there must be a common logical architecture across all switches, routers and network devices to be managed by an SDN controller. This logical architecture can be implemented in different ways, as long as the SDN controller sees a uniform logical switching function across equipment from different vendors and different types of network devices. Second, a secure and standard protocol is needed between the SDN controller and the network device.

2.1 SDN networks scenario

OpenFlow[18] covers these two requirements: it is a protocol between SDN controllers and network devices and allows the logical structure of network switching functions to be specified. The ONF is a consortium of software vendors, content distribution networks and network equipment vendors whose goal is to promote software-defined networking.

According to this documentation, OpenFlow is a protocol that has evolved over time and has undergone some drastic changes between versions as can we see in Figure 3. At the end of 2018, OpenFlow was at version 1.5, but version 1.3 will be taken as a reference to explain the main features of this protocol. As will be seen below, this selected version allows to operate and use the devices without inconvenience. However, it is advisable to know the features of future new versions of OpenFlow, since these will most likely be the ones implemented in most of the switches.

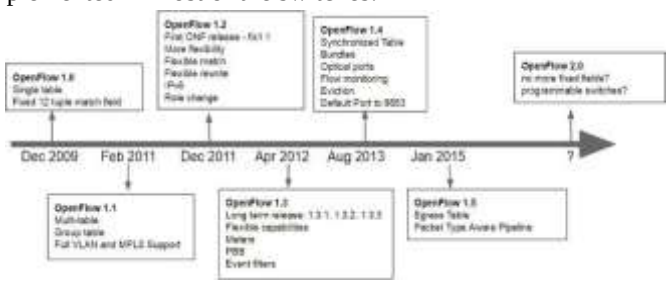

Figure 3. OpenFlow version history[18]. Standardizations in SDN.

Initially, due to the great success of telecommunication technologies, various applications with divergent requirements for networks have been developed. To meet these requirements, networks need to become even more controllable and manageable. The need to manage traffic in different ways implies an increase in service orientation.

Numerous emerging packets forwarding technologies are enabling more direct, lower-level data control methods, e.g., flow level. These technologies can simplify interaction with network resources (switches, routers, etc.) to significantly increase network control capability. Centralized automation of the modeling and scheduling of network resources will enable much more agile network operation. This centralized, programmable approach can provide the opportunity to redesign network resource control functions using standard interfaces and protocols.

Therefore, we can say that this approach allows:

- Centralized logical control of the network, reducing the number of points to control and manage,
- Support for network virtualization as an important feature of the network architecture and
- The definition, control and management of network resources through software applications, providing network services in a deterministic manner according to behavioral requests and customizing the network for efficient and effective deployment in its operations.

To implement the above features, ITU-T Y.3300 provides the framework for software-defined networking (SDN) by specifying definitions, objectives, high-level capabilities, requirements and high-level architecture fundamental to SDN.

Several SDN-related technologies and standards have been developed with different approaches such as ITU-T Y.3001[19], ITU-T Y.3011[20], b-ITU-T Y.2622[21], b-ETSI NFV[22], b-IETF I2RS[23], b-IETF RFC 3746[24], b-ONF[25] and b-OpenDayLight[26]. All share the same goal of providing programmability of network resources which, as mentioned, is a core technology for the networks of the future.

2.2 Flow tables

As mentioned above, flows are grouped into tables similar to routing or switching tables. When a packet is received, each element in the table is checked for compliance with the matching requirements. In OpenFlow version 1.0 there was only one flow table but from version 1.1 onwards nested flow tables are supported as shown in Figure 4. OpenFlow follows a clearly defined process for traversing the flow tables in the switch. This process, or pipeline, is as follows:

- The flow tables in the switch are sorted by numbers, starting from 0.

- Table 0 must always exist since there must be at least one flow table in the switch. The process always starts in this first table.
- When a packet enters, an attempt is made to match to an entry in table 0. If a match is found, the instructions associated with that flow are added to the packet's action set. The action set is the set of activities that are applied to the incoming packet after traversing the flow tables.
- If the input instructions include the instruction to advance to another table ("go to" instruction), the process continues on that table and then repeats the same process. Note that you must always advance to tables with higher numbers, never lower. You must always advance forward, and you can never go back to previous tables.
- When there are no more tables to traverse, i.e., when the match of a table does not include the "go to" instruction, the action set associated with the packet is executed in the preset order.

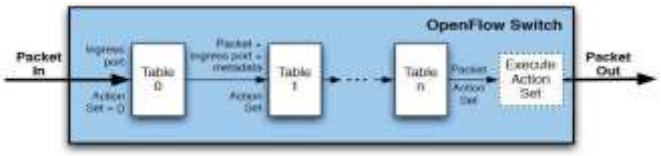

Figure 4. OpenFlow 1.3 Pipeline[27].

If there is no entry in the table associated with the incoming packet, a "table miss" occurs. In this case, the behavior will depend on the table configuration. To indicate how to process mismatched packets, specific entries can be added to the table. The options are a) send the packet to the controller for routing, b) pass the packet to another table in the switch, or c) discard the packet.

Note that the order of the flow entries indicates their priority, similar to firewall rules or router access lists. This implies that two flow table entries could be associated with the same incoming packet. However, when the table is traversed in descending order, only the instructions associated with the first matching entry will be executed.

In OpenFlow 1.5, the pipeline is modified as shown in Figure 5. So far, we have seen the Ingress Processing. Now we will add the Egress Processing. Fundamentally, egress processing follows the same operation as ingress; the tables are traversed looking for valid matches for the package and the instructions of the table associated to the input are executed. Egress processing allows for greater granularity and organization of the flow table entries. It is an optional processing, and it will not be necessary for the switch to implement it in order to route traffic correctly.

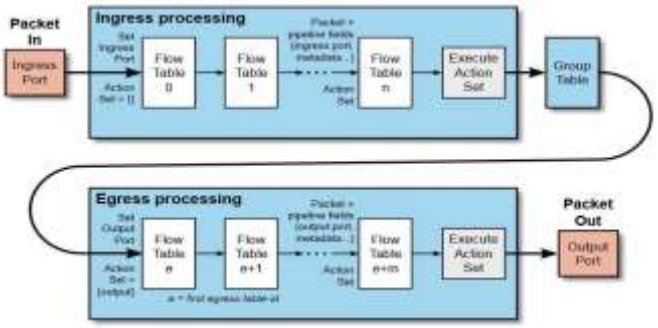

Figure 5. OpenFlow 1.5 Pipeline[27].

2.3 Instruction and action sets

As mentioned, each incoming packet has an action set associated and each flow table entry contains a set of instructions. It is important to understand this distinction. The action set associated with an incoming packet is executed over the packet at the end of the table traversal. In contrast, the instructions associated with a flow entry in the table are executed when an incoming packet matches that entry.

First, we will deal with instructions. There are three types of instructions depending on the task to be performed:

- Advance in the pipeline (advance to another table).
- Modify the action set of the package.
- Modify the incoming packet without waiting for the execution of the final action set of the pipeline (optional).

In total there are six instructions; two are mandatory and four are optional. The switches must necessarily provide support for the mandatory ones. As for the optional ones, the controller can query the switches which of the optional instructions they can support themselves. The mandatory instructions are as follows:

- *Write-actions <action/s>*: Inserts one or more actions into the action set of the incoming package. If any of the inserted actions are already in the action set, they are overwritten.
- *Goto-table <ID of the table>*: Indicates the next table to follow.

Optional instructions are as follows:

- *Meter <id metric>*: Applies a constraint to the specified metric.
- *Apply-Actions <action/s>*: The specified actions are applied immediately on the package without waiting the execution of the action set.
- *Clear-Actions <action/s>*: Removes all actions contained in the action set.
- *Write-Metadata <metadata/mask>*: Updates the metadata.

Each entry in the flow table can only associate one instruction of each type and they will be executed in the

defined order. In practice and for mandatory instructions, this implies that "Goto-table" always goes after "Write-actions". Otherwise, the process will follow to the next table before executing the actions in the action set of the incoming packet.

The action sets are executed over the package at the end of the pipeline as mentioned above and the actions are added to the action set as the successive flow tables are traversed. Actions are not executed on a first-come, first-served basis. The order of execution (priority) of actions is defined as follows:

- *Copy TTL inwards*: Copies the TTL field of the outermost IP header of the incoming packet to the second outermost one with a TTL field. This action is executed when there is an IP header nested inside an MPLS header (which has TTL).
- *Pop*: Removes all VLAN, PBB and MPLS tags from the packet.
- *Push-MPLS*: Adds an MPLS label to the packet.
- *Push-PBB*: Adds a PBB label to the package.
- *Push-VLAN*: Adds a VLAN tag to the packet.
- *Copy TTL outwards*: Copies the TTL field of the second outermost IP header of the incoming packet with TTL field to the outermost IP header.
- *Decrease TTL*: Decreases the TTL of the IP header by one.
- *Set*: Used with optional instructions.
- *Qos*: Applies Quality of Service actions.
- *Group*: Used with optional instructions. Its operation will not be discussed in detail in the present work.
- *Output*: Sends the packet through the specified port.

2.4 Virtualization

A key advantage of SDN technology is the ability to offer network administrators the ability to write programs that use SDN APIs to control network behavior. SDN allows users to develop network-aware applications, intelligently monitor network conditions and automatically adapt network settings as needed[28].

In fact, SDN is an approach to network virtualization that seeks to optimize network resources and quickly adapt networks to changing business, application and traffic needs. It works agilely by separating the network control plane from the data plane, creating a software-programmable infrastructure that is independent of physical devices. With SDN, network orchestration, management, analytics and automation functions become the job of SDN controllers. Because these controllers are not network devices, they can take advantage of the scalability, performance, and modern storage and processing resources of the cloud. Increasingly, SDN controllers are built on open platforms using open standards and open APIs that allow them to orchestrate, manage and control network devices from different vendors.

SDN offers a wide range of business benefits. Separation of the control and transport layers increases flexibility and speeds time to market for new applications. The ability to respond faster to problems and outages improves network availability. For IT organizations, the programmability facilitates the automation of network functions while reducing operating costs.

SDN fits perfectly with another technology: NFV (Network Functions Virtualization). NFV offers the ability to virtualize device-based network functions such as firewalls, load balancers and WAN accelerators. The centralized control provided by SDN can effectively manage and orchestrate this NFV-enabled function virtualization[29].

3. ROUTING IN SDN

At a very high level, the control plane of an SDN establishes the local data set used to create the forwarding table entries. The data plane uses these tables to forward traffic between inbound and outbound ports of the same device.

By centralizing control, all the information available at each switch in the network can be obtained as a single entity. Upon changes and according to the QoS requirements, the software applications will dynamically modify the flow rules taking into account the information of these entities.

3.1 SDN Routing: Architectures for Analysis

The new concept of decoupling these two planes in the network introduced by SDN implies that devices execute only data forwarding; forwarding decisions are based on the set of rules determined by an external controller. This architecture is illustrated in Figure 6[30].

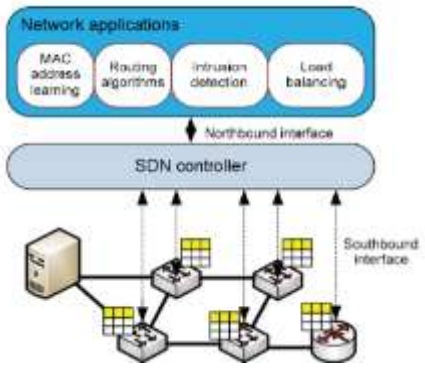

Figure 6. SDN architecture.

For this analysis, we have considered the OpenFlow protocol. This protocol is used for communication between the SDN controller and the data plane devices (OpenFlow switches). OpenFlow enables routing based on decision flow. OpenFlow switches differentiate and process traffic according to instructions received from the controller. In a broad sense, flow could be defined as a sequence of packets with similar

characteristics. As shown in Figure 7, the controller will define the flow using any subset of the 9 L2-L4 packet header fields (layer 2 - layer 4) along with the identifier of the interface the incoming packet. This highly granular control option allows the implementation of dynamic multi-path routing and can significantly increase network capacity[31].

Figure 7. L2-L4 fields used for a flow definition in OpenFlow.

The controller instructions are stored inside the OpenFlow devices in the form of flow table rules (Figure 6). When a packet arrives, the lookup process begins by searching for the corresponding rule in the table. If the packet does not match any rule, it is discarded. Most commonly, however, a low priority rule is defined that instructs the switch to forward the packet to the controller. The SDN controller will then define the next processing steps according to the running network.

In the present case, dynamic routing functionality was implemented within an OpenFlow POX Controller[32]. As shown in Figure 8, the controller manages several key modules: link cost calculation, route calculation, routing, statistics collection, topology discovery, OpenFlow interface, topological data, among others. Some of them are described below.

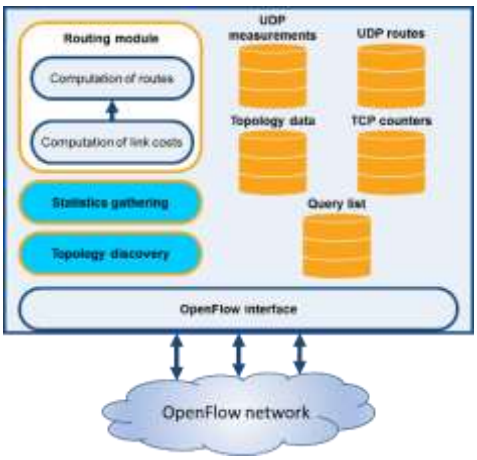

Figure 8. Schematic of the proposed controller design.

- *Topology discovery*: This module is an integral part of the POX controller. It is responsible for discovering and maintaining the network topology. Information about the network connectivity will constantly be available to the other service modules residing in the controller. In case the link goes up or down, this module reports the change events to notify the registered listeners.
- *Statistics gathering*: To perform dynamic routing, the controller needs an up-to-date view of the network status. One option is to measure link utilization periodically. To do this, it will send various port statistics requests to the OpenFlow switches. When an OpenFlow switch receives this request, sends the number of bytes that traversed the corresponding network interface (traffic). By comparing these new values with those previously received, the SDN controller can calculate the link load for the last measurement cycle. As TCP flows tend to use all available bandwidth on the path, this method might conclude that, for example, routing should be avoided on links loaded with one or more long-lived TCP connections. In other words, if the same number of bytes is carried over two different links during the measurement cycle, both links will be treated the same even if there is a considerable difference in the amount of traffic flows on each link.

As a consequence, this will have a negative effect on routing fairness and overall network performance. For this reason, the estimation of the available bandwidth on the links is performed to provide this information to other modules.

Since the links are bidirectional, statistics are compiled for each direction. The reason is based on the fact that TCP tends to distribute the available bandwidth among the individual TCP flows. If the link is assumed to be the bottleneck of each TCP connection, the amount of bandwidth that could be offered to a new TCP flow can be calculated by a formula. As this is a very rough assumption, to determine the actual bandwidth that a new TCP connection could get, the controller needs detailed statistics for each traffic flow on the network.

Estimates of available link bandwidth are only used as input arguments to the routing module. As an example, algorithms can be proposed to protect links with low available bandwidth. On the other hand, when a route for the UDP flow is installed, the controller adds the flow's input switch to the query list (Figure 8).

The statistics collected from the incoming switches provide adequate accuracy, as they take into account packet loss within the network. To minimize overall control, the controller sends only one request for all UDP flows installed on the switch. The response consists of a message with the counters for all incoming UDP flows.

From the statistics obtained, the controller takes the UDP flow throughput information from the last measurement cycle. As illustrated in Figure 8 it also keeps the list of current UDP routes to calculate the UDP throughput on each network link.

The Routing Module calculates the costs of each link and determines the routes for traffic flows. It uses as inputs the network topology data and the results of the statistics collection module.

3.2 Northbound and Southbound

3.2.1 Southbound

In SDN, Southbound interfaces are the OpenFlow protocol specification. They enable communication between controllers, switches and other network nodes that are considered as lower-level components. The router identifies the network topology, determines the network flows and send the request through the Northbound interfaces.

Southbound APIs allow the end user better control over the network and improve in real time the level of efficiency of the SDN controller according to the demands and needs. This interface is an industry standard that justifies the ideal approach that the SDN controller should communicate with the forwarding plane to modify networks by progressively accompanying advanced business needs. To make the network layer more responsive to real-time traffic demands, administrators can add or remove entries to the internal flow tables of network switches and routers.

Some of the most popular Southbound APIs are OpenFlow (discussed in the previous section) and Cisco. Other switch and router vendors that support OpenFlow include IBM, Dell, Juniper, Arista, among others.

3.2.2 Northbound

Unlike Southbound APIs, Northbound APIs enable communication between higher-level components of the network. While traditional networks use a firewall or load balancer to control the behavior of the data plane, SDN installs applications on the controller to communicate with these components through its Northbound interface.

Experts say that it would be quite difficult to improve a network infrastructure without a Northbound interface because the evolution of network applications will depend directly on equipment vendors. The Northbound APIs make it easier to innovate or customize SDN network controls because these APIs can be reprogrammed in languages such as Java, Python or Ruby.

3.3 Another approach: Segment routing

Segment Routing is a flexible and scalable form of routing at the source. The source chooses a path and encodes it in the packet header as an ordered list of segments. Any type of instruction can identify these segments. Each segment is identified by the segment ID (SID) which consists of a 32-bit unsigned integer.

With Segment Routing, the network no longer needs to maintain per-application and per-flow state. Instead, it executes the forwarding instructions provided in the packet. Segment Routing is based on a small number of Cisco extensions and the OSPF (Open Shortest Path First) and IS-IS (IntermediateSystem-to-IntermediateSystem) protocols. It can operate with MPLS or an IPv6 data plane and integrates with MPLS multi-service capabilities including L3VPN (Layer 3 VPN), VPWS (virtual private wire service), VPLS (virtual private LAN service) and EVPN (Ethernet VPN).

Segment Routing can be applied directly to MPLS architecture without changes to the forwarding plane. It uses network bandwidth more effectively than traditional MPLS networks and offers lower latency. A segment is encoded as an MPLS label. The list of segments is encoded as a label stack. The segment to process is at the top of the stack. The segment related to the label is removed from the stack after completion.

Segment Routing can be applied to IPv6 architecture with a new type of routing extension header. The segment is encoded as an IPv6 address. An ordered list of segments is encoded as an ordered list of IPv6 addresses in the routing extension header. The segment to process is indicated by a pointer to the route in the extension header. The pointer is incremented after the completion of a segment.

Segment Routing provides automatic traffic protection without topological restrictions. The network protects traffic against link and node failures without requiring additional signaling. FRR (Fast IP routing) technology in combination with Segment Routing ensure full protection with optimal backup paths. Traffic protection imposes no additional signaling requirements[33].

Segment Routing is SDN-ready: Segment Routing is a compelling architecture designed to be adopted by SDN and is the foundation of AER (Application Engineered Routing). It strikes a balance between distributed network-based intelligence such as automatic link with node protection and centralized controller-based intelligence such as traffic optimization. It can provide a network with performance guarantees, efficient use of network resources and very high scalability for applications.

The network uses a minimal amount of state information to meet these requirements. Segment routing can be easily integrated with controller based SDN architectures. Figure 9 illustrates an SDN scenario in which the controller performs centralized optimization including bandwidth admission control. In this scenario, the controller has a complete view of the network topology and flows. A router can request a route to a destination with certain characteristics such as delay, bandwidth, diversity, among others.

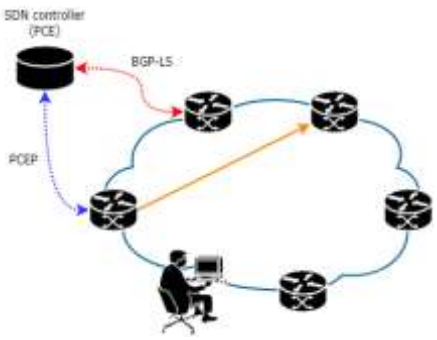

Figure 9. Source Cisco Systems

Segment lists allow complete virtualization of the network without adding any application state. The state is encoded in

the packet as a list of segments. The network maintains the state of a large number of segments and can support a large number of transaction and request-based applications without increasing the network traffic load.

3.4 The need for an interface to enable application programming - I2RS[34]

One of the problems of centralizing control planes is to be able to configure and obtain statistics from network devices in a synchronized manner and in real time. As this process is slow today, efforts are being made to speed up the feedback loop with the network elements through the I2RS project (RFC 7921)[35][36][37].

The I2RS Working Group was formed in November 2012 to develop the use cases and basic architecture of a routing system interface. The term "routing system" describes a hardware device, a virtual router, or any software that provides routing functions[38].

- The routing system interface shall provide the following functionalities:
- Provide or retrieve information, policies and operational parameters to or from the routing system.
- Provide read and write access to the RIB (Routing Information Base) but not to the FIB (Forwarding Information Base).
- Monitor and analyze BGP operations and establish or activate protocol-related policies.
- Optimize and choose network egress points based on factors other than those provided by routing protocols.
- Support rapid and distributed reaction to network attacks.
- Redirect traffic to avoid the destination under attack while maintaining normal operation of other routes.
- Extract network topology information.

Routers that belong to the Internet routing infrastructure maintain details and state functions in a multilayer fashion. For example, a typical router maintains a RIB and implements routing protocols such as OSPF, IS-IS and BGP to exchange reachability information, topology, protocol state, etc. with other routers. Routers convert all this information into forwarding entries that are then used to forward packets and flows between network elements.

It was mentioned earlier how an SDN treats packets. It is noted that flow table rules may require network topology information from the RIB and device configuration.

The forwarding plane and forwarding entries contain information needed by network-oriented applications regarding the expected and observed operational behavior of the router. Network-oriented applications require easy access to this information to know the network topology, to verify that the expected configuration is installed in the forwarding plane, to measure the behavior of flows, routes or forwarding entries, as well as to know the configured and active states of the router. These applications also require an easily accessible interface to program and control the state of the forwarding plane.

I2RS facilitates the control and observation of routing states and allows network-oriented applications to interact with existing networks. These applications can leverage I2RS as a programmatic interface to create new ways to combine the retrieval and analysis of Internet routing data and state configuration within routers.

I2RS provides a framework for applications (including those of the controller) to record and request specific information that each may need or require.

There are four key drivers that shape the I2RS architecture:

1. The need for an asynchronous and programmable interface for all operations to have quick and interactive access.
2. Access to structured information and state that is not modeled or configurable in existing configuration protocols.
3. The ability to subscribe to structured event notifications from the router.
4. The availability of standard data models for network-oriented applications.

When a router or the I2RS software running on it is restarted it will restart in an initial ephemeral state. The routing protocol or application will be able to inject a certain state into the router through the state insertion functions of I2RS. This state can then be distributed by a routing or signaling protocol so that it can be used locally (e.g., to schedule the shared forwarding plane).

A local client operates in the same physical container as the routing system, whereas a remote client operates throughout the network. The details of how applications communicate with a remote client is beyond the scope of I2RS.

I2RS agents and clients communicate with each other using an asynchronous protocol where a single client can publish multiple simultaneous requests to one or multiple agents. An agent can process multiple simultaneous requests from one or multiple clients.

The I2RS agent provides read and write access to selected data from any routing element available to I2RS services.

I2RS agents can write static ephemeral state (e.g., RIB entries) and read static and dynamic routes (e.g., MPLS label switched path identifier or the number of active BGP neighbors). The I2RS agent also allows clients to subscribe to different event notifications that affect different instances of the objects, e.g., notification of an event that resolves the next hop in the RIB such that a RIB manager can install it in the forwarding plane as part of a particular route.

The I2RS agent can access and modify the data models associated with the routing system that enable dynamic, static, local, routing and signaling configuration.

Routing elements need not have an associated forwarding plane to implement some subset of the routing system. Examples of routing elements may include:

- A router with a forwarding plane and RIB Manager that runs IS-IS, OSPF (Open Short Path First), BGP (Border Gateway Protocol), PIM (Protocol Independent Multicast), etc.
- A routing element can be managed locally via the CLI (Command Line Interface), SNMP or via NETCONF (Network Configuration Protocol).

The routing and signaling module that interacts with the Internet routing system is part of the routing element. It includes not only the standardized protocols like IS-IS, OSPF, BGP, PIM, RSVP-TE, LDP (Lightweight Distribution Protocolo), but also the RIB management layer.

To achieve dynamic routing, an I2RS agent needs access to the state of the routing element beyond the subsystem because it may need information from counters, statistics, flows and local events. I2RS-based network applications need this operational state, but I2RS does not provide this standardized routing and signaling information. An example of a static system state is the queuing behavior for an interface or for traffic. How the I2RS agent modifies or obtains this information is beyond the scope of this paper.

Recently, many efforts have been made to improve access to existing information in routing and forwarding systems. The greatest benefits have been achieved by making network information visible and manageable through applications. For this, there are two major challenges; first, to consider the large amount and variety of information potentially available and second, to simplify the protocol for accessing such information due to the complexity in the variation of data structures and the types of operations required.

The types of operations contemplated are complex in nature. It is essential that I2RS be easily implementable and robust. Complex data models make extensibility difficult, so I2RS does not attempt to add more complexity than necessary.

A network topology manager includes an I2RS client that uses its own data models and a protocol that collects network state information by communicating with one or more I2RS agents. The topology manager collects routing configuration and operational data, such as interface and switched path label (LSP) information. In addition, the topology manager can collect link-state data in several ways: through I2RS models, with BGP-LS (RFC 7752)[39], or by listening to the IGP.

The set of functionalities and information collected by the topology manager can be integrated as a component of an application, such as a route calculation application. Other application interfaces can provide a consistent view of the network reachability state using the same I2RS protocol or could provide a topology service using extensions of the I2RS data models. To perform this task, the controlling entity or program has to generate a network topology view under certain conditions. This network view can be manually programmed, learned through observation, or constructed from information gathered through exchange with other control plane records.

In the case of SDN, all this information to be processed and transmitted belongs to the so-called control plane. In this context, the control plane can be defined as the intelligence that determines the optimal paths for sending information and responds to incidents and new network demands[40].

In summary, the basic idea of I2RS is to create a protocol and components for scheduling the routing information base (RIB) of a network device. It uses a fast-routing protocol that allows a fast cut-through of provisioning operations in order to allow real-time interaction between the RIB and the RIB controller manager. Previously, RIB could access information only through the device configuration system such as Juniper[41], Netconf[42] or SNMP[43].

I2RS provides several levels of abstraction for network path scheduling, policy management and port configuration, e.g., to provide fast and optimal access to the RIB for operational support systems (OSS)[44].

I2RS is also well oriented towards possible growth in the requirements for logical centralization of routing, routing decisions and programmability. The protocol has different requirements for running inside or outside a device. In this way, as required, the functionality of a distributed controller can be implemented.

Finally, another key subcomponent of I2RS is the standardized and abstract topology. This topology will be represented by common and extensible object models. The service also allows multiple abstractions of a topology representation to be exposed. A key aspect of this model is that devices that are not routers (or routers of routing protocols) can more easily manipulate and change the RIB. Today, users do not have a great deal of difficulty obtaining this information. In the future, network management/OSS components will be able to quickly and efficiently interact with routing state and network topology.

3.5 Interaction between SDN and existing networks in use

The "clean slate" proposition is a statement related to SDN. SDN discards previously used technologies by thinking of operating mechanisms outside the distributed model, thus avoiding the technological costs due to the complexity of its adaptation.

This proposition arises from observing the MPLS technology where the updates and modifications of its features made the deployed code of the implementations grow, transforming them into too complex and fragile.

In some implementations[45], using centralized label distribution to emulate the distributed functionality of LDP[46] or RSVP[47], centralized code-based network topology

knowledge yielded results at least an order of magnitude lower than currently available commercial alternatives. The natural assertion is that, in a highly prescriptive and centralized control system, network behavior can approximate that of a completely static forwarder that is arguably stable.

When network utilization increases, new devices must be deployed to meet the demand. It would be important to be able to meet the forwarding transfer demand without changing the number of managed devices and their resulting control protocol entities; if not, at least determine the integration conditions of the new devices since each addition increases the control information.

If we think that our centralized system must interact with distributed devices typical of traditional routing and switching systems, it is important to understand that the traditional controller and router control planes must be synchronized to achieve information consistency. The additional control plane affects the scalability of the overall network control plane during network convergence, or the time it takes for the totality of control planes to reach a loop-free circuit. This manifests itself in the resilience and performance of the overall system; the greater the number of control planes there will potentially be greater fragility in the system. Conversely, if properly tuned, the resilience of the system will increase due to creating a system that becomes consistent regardless of conditions.

It seems inevitable to discard the initial idea of the "clean slate" as the current networks with their routing protocols will continue to exist. These legacy networks will have to interact with SDN networks in a mixed topology as seen in the I2RS work discussed above.

4. EXAMPLE OF ROUTING IN A SWITCH

In order to better understand the scope of SDN, the following is a simple example of how routes are established in a switch and how the interaction with the controller is performed.

This example presents a network topology that has two switches SW1 and SW2 with 2 hosts h1 and h2 connected to each switch respectively, as shown in Figure 10.

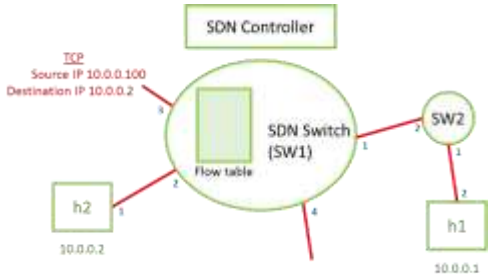

Figure 10. Example topology.

We will analyze the behavior of the SW1 which is connected to other network devices through ports 1, 3 and 4 and to h2 through port 2. Inside the SW1, there is the flow table with the actions to be applied to the incoming packets according to the matching criteria for each input.

The criteria can be defined on the basis of fields in the Ethernet, IP, TCP, UDP header or any other protocol that contains a packet header. In this case, the source and destination IP addresses will be used as matching criteria. Those packets matching these criteria will be subjected to the actions programmed in the table such as forwarding to a SW1 port, forwarding to the controller to modify incomplete fields in the packet or other actions.

Packets arriving on port 2 or 3 will be directed to host h1 (10.0.0.1) by the corresponding actions: they will exit on port 1 of SW1, traverse SW2 and then exit on port 1 of SW2 to reach h1. Packets arriving with destination 10.0.0.2 will also be instructed to be dispatched to host h2 on port 2 of SW1. In these cases, no conditions are set that consider the source address of the packets, nor the transport protocol.

A third entry will be added so that TCP packets entering SW1 will be dispatched on port 4 regardless of their source or destination IP addresses. You get a Table 1 like the one shown below:

Table 1 - SW1 flow table.

FLOW TABLE						
Pairing criteria			Actions	Priority	Duration	Idle timeout
Source IP	Destination IP	Protocol				
*	10.0.0.1	*	out: 1	2	20 sec	5 sec
*	10.0.0.2	*	out: 2	2	20 sec	5 sec
*	*	TCP	out: 4	1	0 sec	5 sec

If you look at the first two entries in Table 1 you will notice that packets without TCP protocol information will be routed to hosts h1 or h2 through ports 1 and 2 respectively.

If you enter a TCP packet with source IP address 10.0.0.100 and destination IP address 10.0.0.2, SW1 would not be able to decide whether to route it to port 2 or 4 without the information in the Priority field. In these cases, it is routed to the output with the higher value, on port 2.

Finally, there are two fields that establish times. The first one indicates the duration of the rule from the time it was installed by the controller, i.e., the time it remains in the table. The second indicates that the rule will be removed if no matching takes place during this period, i.e., no packet arrives that complies with the rule.

Placing 0 (zero) in the "Duration" and "Inactive timeout" fields will indicate that the rule can only be removed by the controller.

The controller will control the switches via a TCP connection that is established at the time of installation. The controller will have applications installed that will react appropriately when recognizing certain events. Examples are given below.

We will add a new SW3 switch to the initial topology. A TCP connection will be established between the controller and the new switch. Then, from each side of the connection, the

switch and the controller will generate a symmetric OFPT_HELLO[48][49] message that establishes the OpenFlow connection and modifies the topology known to the controller as will be seen later.

The addition of SW3 will generate changes in the SW1 switch. The controller will modify the SW1 flow table through OFPFlowMod messages creating the specific entries, indicating the match criteria, action priority and associated times. Figure 11 shows in detail the event produced by incorporating the SW3 switch.

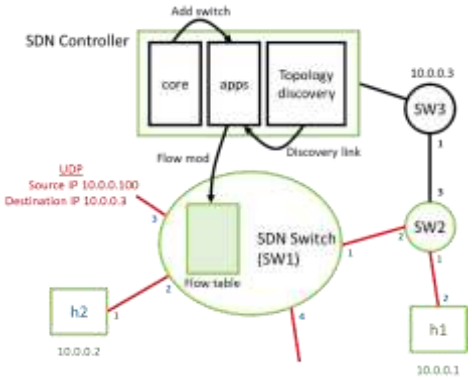

Figure 11. Node incorporation event.

When the packet does not match any entry in the table, the packet or its header will be sent to the controller by OFPT_PACKET_IN message. The central module of the controller will have an application to resolve the sending of these packets. This application will determine the flow rules to be installed in SW1 using OFPFlowMod messages, as shown in Figure 12.

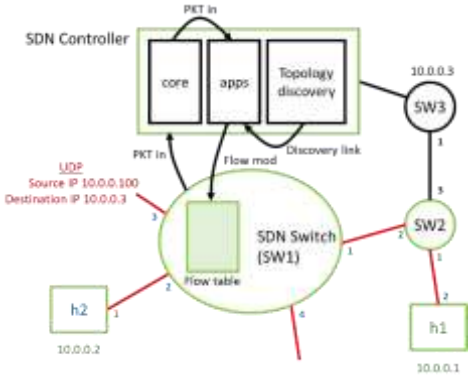

Figure 12. Packet event not matching any table entry.

Finally, when a link is discovered, the topology discovery module of the controller changes the network topology by exchanging discovery packets. In our example, this discovery event is the addition of SW3. The topology discovery application must be able to respond by generating the necessary OFPFlowMod messages to add or remove the necessary actions in the corresponding flow tables.

5. CONCLUSIONS

- In modern routers and switches, configuration changes will affect the results obtained in the control and data planes. Centralized management will impact the behavior and results in distributed nodes.
- For many network administrators, control is based on the flexibility to program simple forwarding decisions. Simple and specific solutions will be favored over complex ones.
- It will be very important to make the network more elastic and efficient based on the demands and new information available beyond optimal algorithms.
- Both static routes and routing policies have limited scalability in most implementations.
- Destination-based distributed routing protocols are not a good solution in the application of pure centralized routing ("clean slate"). SDN networks should be interconnected with traditional networks that maintain standardized routing protocols that should be part of the information stored in our controller's RIB.
- The idea of routing based on optimal path finding seems to give way to balanced routing that meets compromised QoS parameters for all flows transmitted over the network using all its available links and resources.
- Networks will be able to adapt to changes by using metrics to control available bandwidth, packets lost from prioritized flows or required jitter. Applications will be used to dynamically modify the rules in the flow tables, either through static programming or dynamically following behavioral patterns.
- SDN networks have an intrinsic characteristic: any type of algorithm (such as Dijkstra's) can be applied without being tied to routing protocols (such as OSPF). Therefore, and depending on the rules that are established, it will be possible to arbitrarily alter proprietary algorithms and third-party algorithm libraries.
- It is clear that the key to SDN will be to develop communications networks where the control plane is decoupled from the hardware elements. Unlike traditional networks, the centralization of functions in the controller will simplify network traffic management tasks without having to configure the elements individually.

REFERENCES

1. IEEE Xplore Digital Library – https://ieeexplore.ieee.org/Xplore/home.jsp - Accessed: 18-11-2021.

2. L. Pu W. Wu C. Akyildiz, f. Ahyoung. A roadmap for traffic engineering in SDN-OpenFlow networks. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=73DCA07DA08783DAB984C94FA3559DBE?doi=10.1.1.433.1760&rep=rep1&type=pdf - Accessed: 18-11-2021.
3. Esteban Carisimo, Dr. Ing. J.Ignacio Alvarez-Hamelin ,VII National Meeting of Technicians, ArNOG - CoNexDat - INTECIN (UBA-CONICET), Buenos Aires, Argentina- https://cnet.fi.uba.ar/esteban_carisimo/presentaciones/ArNOG_2017_1.pdf - Accessed: 14-02-2020.
4. What is an SLA? How to Use Service-Level Agreements for Success - https://www.process.st/sla-service-level-agreement/ - Accessed: 24-02-2020
5. RSVP / Intserv and Diffserv Network Interoperation - March 1999. – https://datatracker.ietf.org/doc/html/draft-ietf-issll-diffserv-rsvp-01 - Accessed: 14-02-2020
6. Traffic forwarding in a network with IPQoS: Hopping behaviors https://docs.oracle.com/cd/E19957-01/820-2981/ipqos-intro-54/index.html - Accessed: 24-02-2020.
7. Inter-domain Traffic Conditioning Agreement (TCA) Exchange Attribute – https://datatracker.ietf.org/doc/draft-ietf-idr-sla-exchange/ - Accessed: 24-02-2020.
8. RSVP / Intserv and Diffserv Network Interoperation - March 1999. https://datatracker.ietf.org/doc/html/draft-ietf-issll-diffserv-rsvp-01 - Accessed: 14-02-2020.
9. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Header - https://tools.ietf.org/html/rfc2474 - Accessed: 14-02-2020.
10. Cisco - Implementing Quality of Service - https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-packet-marking/13747-wantqos.html - Accessed: 18-11-2021.
11. S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken - The nature of data center traffic: Measurements & analysis, in: Proc. ACM SIGCOMM IMC, Chicago, Illinois, USA, 2009.
12. A Load Balancing Method Based on SDN. Author(s) Mao Qilin ; Shen Weikang - 7th International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2015. https://ieeexplore.ieee.org/document/7263504 - Accessed: 18-11-2021.
13. Dynamic Load Balancing Application For Servers, Based On Software Defined Networks. Iberian Journal of Information Systems and Technologies. Carlos Enrique MINDA GILCES1, Rubén PACHECO VILLAMAR2. – http://www.scielo.mec.pt/pdf/rist/n32/n32a06.pdf - Accessed: 14-02-2020
14. Jack Tsai, Tim Moors. A Review of Multipath Routing Protocols: From Wireless Ad Hoc to Mesh Networks. National ICT Australia / University of New South Wales, Australia, 2006.
15. Software-Defined Networks and OpenFlow - The Internet Protocol Journal, Volume 16, No. 1 - March 2013 - William Stallings - https://paperzz.com/doc/7523373/download-pdf-file--volume-16--no.-1--1.83mb - Accessed: 18-11-2021.
16. What are Operational Support Systems (OSS) and BSS in Telecom?- Passionate about us- https://passionateaboutoss.com/background/whatare-oss-bss/ - Accessed: 08-06-2023.
17. What is SaaS? - Microsoft Co. - https://azure.microsoft.com/es-es/overview/what-is-saas/?cdn=disable - Accessed: 18-11-2021
18. Ching-Hao, Chang and Dr. Ying-Dar Lin - OpenFlow Version Roadmap, 2015 - http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep_frank.pdf - 2011-05-20 - Accessed: 11-06-2019.
19. Y.3001 : Future networks: Objectives and design goals - https://www.itu.int/rec/T-REC-Y.3001-201105-I/en - Approved in 2011-05-20 - Accessed: 26-07-2020.
20. Y.3011 : Framework of network virtualization for future networks - https://www.itu.int/rec/T-REC-Y.3011-201201-I - Approved in 2012-01-13 - ITU - Accessed: 26-07-2020.
21. Architecture of an independent scalable control plane in future packet based networks - Recommendation ITU-T Y.2622 – https://www.itu.int/rec/dologin_pub.asp?lang=s&id=T-REC-Y.2622-201207-I!!PDF-E&type=items – 07-2012 - ITU - Accessed: 26-07-2020.
22. Network Functions Virtualisation (NFV) Release 4 - https://www.etsi.org/technologies/nfv - ETSI - 2019 - Accessed: 26-07-2020.
23. Interface to the Routing System (I2RS) Security-Related Requirements – https://datatracker.ietf.org/doc/html/rfc8241 - Internet Engineering Task Force (IETF) - 09-2017 - Accessed: 26-07-2020.
24. Forwarding and Control Element Separation (ForCES) Framework RFC 3746 – https://datatracker.ietf.org/doc/rfc3746/ - IETF - 01-05-2018 - Accessed: 26-07-2020.
25. Open Network Foundation Website - https://opennetworking.org/ - ONF - Accessed: 26-07-2020.

26. Open Daylight Website – https://www.opendaylight.org/ - Open Daylight - Accessed: 26-07-2020.
27. OpenFlow - Basic Concepts and Theory - David Varnum - https://overlaid.net/2017/02/15/openflow-basic-concepts-and-theory/ - Accessed: 07-03-2023.
28. What is SDN? – https://www.ciena.com/insights/what-is/What-Is-SDN.html - Accessed: 18-11-2021?
29. What is SDN? - Juniper – https://www.juniper.net/us/en/solutions/sdn/what-is-sdn/ - Accessed: 18-11-2021
30. D. Kreutz, F. M. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S, Azodolmolky, S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE, Vol.103, No.1, pp.14-76, Jan.2015. - Accessed: 26-07-2020
31. S. Tomovic, N. Prasad, I. Radusinovic, "SDN control framework for QoS provisioning", 22nd Telecommunication Forum TELFOR 2014, pp. 111-114, Belgrade, Serbia, November 2014.
32. Using the POX SDN Controller. Available: https://www.brianlinkletter.com/2015/04/using-the-pox-sdn-controller/ - Accessed: 07-03-2023
33. Introduction to Segment Routing - https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/seg_routing/configuration/xe-3s/segrt-xe-3s-book/intro-seg-routing.pdf - Accessed: 26-07-2020.
34. An Architecture for Interfacing with Routing System (I2RS)RFC 7921 - https://tools.ietf.org/html/rfc7921 - Accessed: 24-02-2020.
35. S. Tomovic, N. Prasad, I. Radusinovic, "SDN control framework for QoS provisioning", 22nd Telecommunication Forum TELFOR 2014, pp. 111-114, Belgrade, Serbia, November 2014.
36. ASICs: Only the hard facts count - https://www.elmos.com/produkte/special-projects/asic-design.html - Accessed: 18-11-2021.
37. D. Kreutz, F. M. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S, Azodolmolky, S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", Proceedings of the IEEE, Vol.103, No.1, pp.14-76, Jan.2015. - Accessed: 26-07-2020
38. I2RS Architecture in SDN and its Use Cases - Arora, Anshul – 29-09-2021 – https://era.library.ualberta.ca/items/df8dce41-ff58-41d6-b934-d9a797421d6f - Accessed: 08-06-2023.
39. North-Bound Distribution of Link-State and Traffic Engineering (TE) - Information Using BGP - https://tools.ietf.org/html/rfc7752 - Accessed: 24-02-2020.
40. Technopedia, Control Plane. – https://www.techopedia.com/definition/32317/control-plane - Accessed: 31-07-2019
41. Juniper - About Us – https://ejuniper.com/es/juniper/sobre-nosotros/ - Accessed: 18-11-2021.
42. NetConf - RFC 6241 – https://tools.ietf.org/html/rfc6241 - Accessed: 18-11-2021.
43. SNMP - RFC 1157 – https://tools.ietf.org/html/rfc1157 - Accessed: 18-11-2021.
44. Operations and business support systems with Red Hat – 19-03-2020 – https://www.redhat.com/en/resources/oss-bss-streamline-digital-ops-brief - Accessed: 08-06-2023.
45. Dynamic Load Balancing Application for Servers, Based on Software Defined Networking - RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação versão impressa ISSN 1646-9895 ISTI no.32 Porto - jun. 2019. - http://dx.doi.org/10.17013/risti.n32.67-82 - Authors: Carlos Enrique Minda Gilces1, Rubén Pacheco Villamar2 - Accessed: 24-02-2020
46. LDP Specification - RFC 5036 – https://tools.ietf.org/html/rfc5036 - Accessed: 18-11-2021.
47. Resource ReSerVation Protocol (RSVP) - RFC 2205 - https://tools.ietf.org/html/rfc2205 - Accessed: 11-18-2021.
48. Implementation Of An OpenFlow Communications Module For Smartnet. - Douglas Alexander Aguacía Fiscó - Pontificia Universidad Javeriana Facultad De Ingeniería Electrónica Departamento De Ingeniería Electrónica Bogotá -2014. - https://repository.javeriana.edu.co/bitstream/handle/10554/16509/AguaciaFiscoDouglasAlexander2015.pdf?sequence=1&isAllowed=y - Accessed: 16-02-2020
49. OpenFlow v1.3 Messages and Structures - https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html - Accessed: 16-02-2020.

NOMENCLATURE

AER	Application Engineered Routing
API	Application Programming Interface
apps	Applications
BGP-LS	Border Gateway Protocol - Link State
BSS	Business Support Systems
CLI	Command Line Interface
DiffServ	Differentiated Services
DSCP	Differentiated Services Code Point

Dst	Destination
EVPN	Ethernet VPN
FIB	Forwarding Information Base
FRR	Fast IP Routing
HTTP	Hyper Text Transfer Protocol
hx	Host number x
I2RS	Interface to Routing System
IBM	International Business Machines Corporation
ID	Identification number
IEEE	Institute of Electrical and Electronics Engineers
IntServ	Integrated Services
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IS-IS	IntermediateSystem-to-IntermediateSystem
IT	Information Technology
L2-L4	Layer 2 - Layer 4
L3VPN	Layer 3 VPN
LDP	Label Distribution Protocol
MAC	Media Access Control
MPLS	Multiprotocol Label Switching
NETCONF	Network Configuration Protocol
NFV	Network Functions Virtualization
OSPF	Open Shortest Path First
OSS	Operation Support Systems
PBB	Provider Backbone Bridges
PCE	Path Computation Element
PCEP	Path Computation Element Protocol
PHB	Per Hop Behavior
PIM	Protocol Independent Multicast
POX	Python OpenFlow SDN Controller
QoS	Quality of Service
RIB	Routing Information Base
RSVP	Resource Reservation Protocol
rx	Router number x
SaaS	Software as a Service
SDN	Software Developed Networks
sec	Seconds
SID	Segment Identifier
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SWx	Switch number x
TCA	Traffic Conditioning Agreement
TCP	Transmission Control Protocol
TE	Traffic Engineering
TOS	Type of Service
TTL	Time To Live
UDP	User Datagram Protocol