Sebastião Guilherme, E. (2024). Optimization of traffic engineering in the switched ethernet network using a programming algorithm on the protocol per Vlan Spanning Tree +. Gregório Semedo: Science and Development, 1(1), e20
https://revistacdugs.ao/article/view/20

Original Article

REVISTA GREGÓRIO SEMEDO CIÊNCIA E DESENVOLVIMENTO

Optimization of traffic engineering in the switched ethernet network using a programming algorithm on the protocol per Vlan Spanning Tree +

Optimização da engenharia de tráfego na rede comutada ethernet usando algorítimo de programação sobre o protocolo per Vlan Spanning Tree +

Optimización de la ingeniería de tráfico en la red conmutada ethernet usando algoritmo de programación sobre el protocolo per Vlan Spanning Tree+

Sebastian Guilherme Eduardo[1] https://orcid.org/0009-0003-3224-2258

[1]UNITEL, Angola. edsonterreiro@gmail.com

Received: October 26, 2024
Accepted: December 15, 2024

ABSTRACT
Networks currently have a large number of platforms that need to establish communication, requiring efficient planning, as well as an appropriate network design at layer two and layer three levels, taking the OSI model as a reference. Therefore, at layer three level there is traffic engineering with specificity in MPLS network traffic engineering, which provides resources to manage connections and traffic according to the network's needs. However, at layer two level, the Spanning Tree Protocol (STP) and its variants cannot provide the same as layer three, in terms of traffic engineering. Traffic engineering in switched networks is practically non-existent due to the limitation of the STP protocol and its variants due to the impossibility of autonomous traffic balancing. Despite this limitation, programmable networks allow the implementation of scenarios natively limited by the STP protocol, making it possible to optimize traffic engineering in the Ethernet switched network using the Python programming language.

Keywords: Traffic engineering; switched networks; MPLS; Python; STP; OSI.

RESUMO
As redes actualmente possuem um grande número de plataformas que necessitam estabelecer comunicação, obrigando um planeamento eficiente, bem como um desenho de rede adequado a nível da camada dois e camada três tendo como referência o modelo OSI. Assim sendo, a nível da camada três existe a engenharia de tráfego com especificidade na engenharia de tráfego de redes MPLS que proporciona recursos para gerir as ligações e o tráfego em função da necessidade da rede. No entanto, a nível da camada dois, o protocolo *Spanning Tree* (STP) e as suas variantes não conseguem proporcionar o mesmo que a camada três, no que tange engenharia de tráfego. A engenharia de tráfego nas redes comutadas é uma realizada praticamente inexistente em função da limitação do protocolo STP e suas variantes face impossibilidade de balanceamento autônomo de tráfego. Não obstante a esta limitação, as redes programáveis permitem a implementação de cenários nativamente limitados pelo protocolo STP, fazendo com seja possível a optimização da engenharia de tráfego na rede comutada *Ethernet* usando a linguagem de programação em *Python*.

Palavras-chave: Engenharia de tráfego; redes comutadas; MPLS; *Python*; STP; OSI.

RESUMEN
Actualmente las redes cuentan con una gran cantidad de plataformas que definen la comunicación, requiriendo una planificación eficiente, así como un adecuado diseño de red a nivel de capa dos y capa tres tomando como referencia el modelo OSI. Por tanto, en el nivel de capa tres existe una ingeniería de tráfico con especificidad en la ingeniería de tráfico de red MPLS que proporciona recursos para gestionar las llamadas y el tráfico en función de las necesidades de la red. Sin embargo, la capa dos, el Spanning Tree Protocol (STP) y sus variantes no proporcionan lo mismo que la capa tres cuando se trata de ingeniería de tráfico. La ingeniería de tráfico en redes conmutadas es prácticamente inexistente debido a las limitaciones del protocolo STP y sus variantes por la imposibilidad de equilibrar el tráfico de forma independiente. A pesar de estas limitaciones, las redes programables permiten implementar configuraciones limitadas de forma nativa por el protocolo STP, permitiendo optimizar la ingeniería de tráfico en la red Ethernet conmutada utilizando el lenguaje de programación Python.

Palabras clave: Ingeniería de tráfico; redes conmutadas; MPLS; Python; STP; OSI.

INTRODUCTION

The rapid and continued growth of radio access technologies have brought greater complexity to routing and global connectivity for service providers, particularly for cellular mobile telephony.

The advent of 4G/5G RAN services has brought a new routing paradigm, where traffic between interconnected platforms is carried over a packet-switched transport network using the Internet Protocol, IP.

The problem for service providers lies in the existence of several transport networks to provide the routing of 2G/3G, 4G/5G services.

The MPLS protocol comes as a solution capable of providing a convergent transport network, serving as backhaul for the most varied types of traffic, whether oriented towards circuit switching or packet switching.

Inevitably, the switched network appears as a crucial element for interconnecting various platforms and interfaces that allow communication from the access layer to the network core, specifically for mobile phone provider networks, where virtual local area networks (VLANs) transport traffic to the most varied network destinations.

In service provider networks, specifically for cellular mobile networks, the integration of various network access technologies (2G/3G/4G/5G) in the same transport network requires that the radio and core communication interfaces can establish communication over the MPLS network and particularly over the switched network (NEC, 2007).

At layer 3 level, the network provides mechanisms that allow routing loops to be excluded through the Time to Live field of the IP header, as well as managing congestion scenarios through MPLS-TE traffic engineering.

At layer two level, specifically in the Ethernet switched network, the Spanning Tree Protocol (STP) and its variants appear as the only element capable of eliminating layer 2 loops, taking into account that the Ethernet frame structure does not have a field with this specific function (Forouzan, 2010).

However, after the convergence of the switched network has been established, with a loop-free network, the increase in traffic, due to poor network sizing, or even due to the increasing user traffic profile, can lead to congestion scenarios in certain links that interconnect the network.

Per Vlan Spanning Tree

Per Vlan Spanning Tree (PVSTP) operates a separate instance of STP for each Vlan, allowing STP on each Vlan to be configured independently, offering better performance and tuning for specific conditions (Cisco, 2020).

Multiple spanning trees also load balance redundant circuits when they are assigned to different VLANs. Thus, one circuit can forward one set of VLANs, while another redundant circuit can forward a different set (Cisco A., 2020).

Service providers, with specificity for 2G/3G/4G/5G services, currently operate mostly over the IP/MPLS network, as well as over the Ethernet switched network, allowing the connection between the different service interfaces, with each of the interfaces that operate over the IP/MPLS network and over the Ethernet switched network being segmented into VLANs. In this scenario, VLANs allow the partition of a local network into different logical segments, allowing platforms physically connected to different switches to be virtually connected to a single switch. Since VLANs provide high flexibility in a local network, these networks become ideal in environments with various types of traffic that need to be managed (Melo, 2009).

To obtain redundancy at the service level, the interfaces that operate on the layer two network have redundant VLANs, configured on the switches, distributed on the redundant circuits.

If a switched Ethernet network supported the forwarding of 2G/3G service traffic at an average of 1Gbps during peak hour (HMM), the network would have approximately twenty virtual locations and 20Gbps in the network infrastructure.

In a layer two network environment, where PVST is used, STP parameters are assigned so that VLANs are equally distributed across all active links. In this way, there will be VLAN instances on each of the active links, achieving optimal results in relation to load balancing, with one STP instance being maintained for each VLAN. In a scenario with twenty virtual private networks assigned to RAN service networks, we will have twenty instances for different logical topologies, which in principle is considered positive in terms of load balancing, but with a negative impact on the memory and processing of the switch since each VLAN implies the existence of a Spanning Tree instance. Therefore, there was a need to reduce the number of Spanning Tree instances to map a set of VLANs by implementing Multiple Spanning Tree (MSTP) (Edgeworth, Rios, Goley, & Hucaby, 2020).

Multiple Spanning Tree

The Multiple Spanning Tree (MSTP) protocol, defined by the IEEE 802.1.s standard, is an evolution of the RSTP protocol, with the objective of ensuring the existence of several virtual local networks in a reduced number of STP instances, using a virtual local network for control, thus allowing faster convergence and the possibility of load balancing (Melo, 2009).

Each MSTP instance is associated with a set of VLANs belonging to a single administration called a region. Thus, a region is an independent instance of another MSTP, a scenario that allows load balancing of the MSTP instances, allowing traffic from the associated virtual private networks to be forwarded independently of other regions (Vieira, 2010).

The reduction in the number of STP instances and the guarantee of load balancing between links make the MST protocol a choice among layer two protocols in relation to its predecessors. However, the STP protocol, like all variants, does not have enough intelligence to automatically manage congestion scenarios in the layer two network (Edgeworth, Rios, Goley, & Hucaby, 2020).

From the perspective of transmission quality, even if latency values are verified above or below the threshold, packet losses on a given path where the PVSTP+ protocol believes to be the best path to forward traffic, or any other anomalous scenario in the links between transmission trunks, PVSTP will not switch traffic to a link with better transmission requirements autonomously, thus presenting itself as limited in terms of traffic engineering.

Given the flexibility that exists in network infrastructures and with the advent of software-defined networks, the algorithms of the Python programming language allow traffic engineering to be carried out autonomously in the Switched network, over the Per Vlan Spanning Tree + layer two protocol.

Software Defined Networks

Traditional networks are not as flexible as they could be and operational expenses are so high, this is because network environments have often not been able to respond to change requests within the desired time frame (Academy, 2019).

Providers would like to make changes to accommodate their customers' needs, and as they stand today, networks cannot meet these demands. Networks require an injection of software-based control to enable immediate and timely changes to meet these needs (Learning, 2021).

Large network domains used to consist of a manageable number of routers, switches, and other devices, so network engineers could configure and maintain these devices using manual methods. With the increasing load on the network driven by video, voice, social media, service integration over the MPLS protocol (ATOM), storage, and cloud-based applications, the number and complexity of network devices has increased to a point where management must be automated to some degree. Network programmability can provide the capability for efficient device management through software (Learning, 2021).

In a traditional router or switch architecture, control plane and data plane functions occur on the same device. Routing and packet forwarding decisions are the responsibility of the device's operating system (Cisco, 2020).

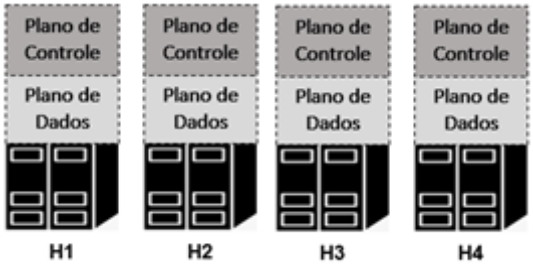

Fig. 1 -Traditional network architecture for traffic forwarding.

Software-defined networks basically consist of the separation of the control plane from the data plane. In this way, the control plane function is removed from each device and is performed by a centralized controller. This centralized controller communicates the control plane functions to each device, where each one can focus on actually sending data, while the centralized controller manages the data flow, increasing security and providing other services (Hadley, Nicol, & Smith, 2017).

The controller is a logical entity that allows network administrators to manage and determine how the data plane of switches and routers should handle network traffic, orchestrating, mediating, and facilitating communication between applications and network elements (Cisco, 2020).

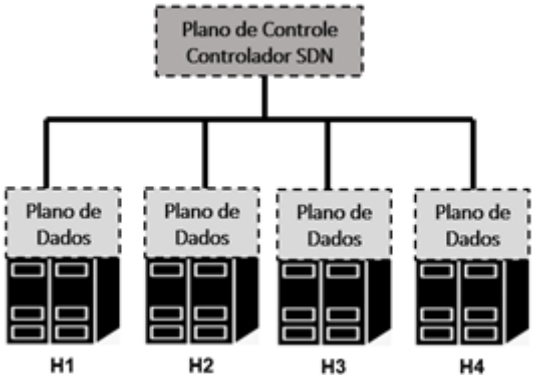

Fig. 2 -Software Defined Network Architecture.

Software-defined networking frameworks use application programming interfaces (APIs), which allow other applications to access their data or services. It is based on the principle of a set of rules that describe how one application can interact with another and the instructions to allow the interaction to occur (Cisco A. , 2020).

Taking into account corporate needs, SDN networks can be device-based, controller-based, or policy-based.

Device-based software-defined networks are programmable by applications and run on the same device.

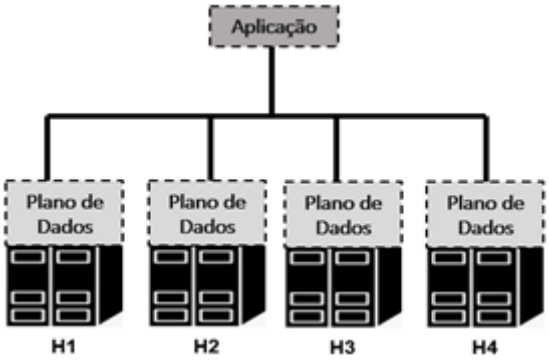

Fig. 3 -Application-based Software Defined Network Architecture.

Controller-based SDNs use centralized controllers that have visibility into all devices on the network, so the application can communicate with the controller that manipulates data flows and manages the devices.

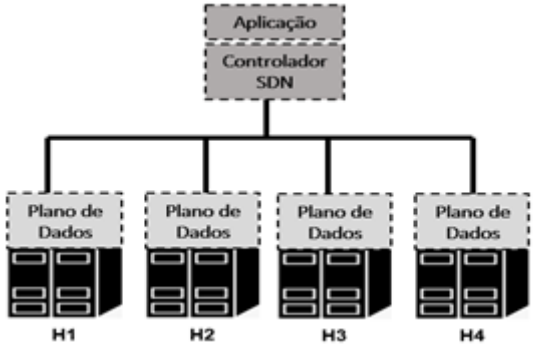

Fig. 4 -Controller-based SDN.

For policy-based SDNs, a central controller has a view of all devices on the network. However, the differentiating element of controller-based SDN is that it uses integrated applications that automate actions on devices through a layer with a higher level of abstraction. It has graphical interfaces that allow the technician to use it without having programming skills.

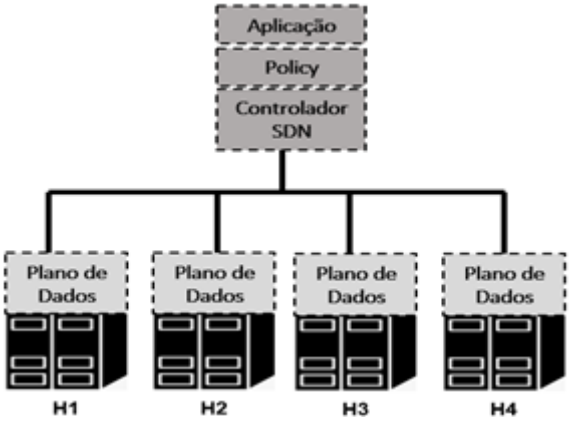

Fig. 5 -Policy-based SDN

Automation in Networks

Network programmability is a trend enhanced and inspired by SDN, based on scripting methods and standard programming languages used for control and monitoring of network elements (Learning, 2021).

The SDN concept seeks to eliminate vendor lock-in through standard protocols such as OpenFlow. However, legacy non-SDN networks need to keep pace and respond to dynamic network changes. Switched network-level protocols such as PVSTP+ exist and are still widely used (Learning, 2021). Despite new protocols and their optimizations, protocols that have been considered legacy will continue to exist in network infrastructures for much longer.

In this scenario, some new methods of configuring network devices using automation appear, reducing equipment configuration time and facilitating maintenance. It is clear that improving network security, recognizing and correcting vulnerabilities, and increasing

network stability represent the future of networks, allowing the management of a greater number of devices in a unitary manner (Sandu & Curpen, 2017).

In an environment where there are multiple network devices and there are no multiple interventions, or there are a reduced number of devices, automation is not necessary. However, the large number of tasks in repetitive scenarios directs the use of scripts to automate network functions (Buresh, et al., 2017).

In addition to automating network functions, more items can be added so that the automation process effectively assumes a crucial role, among which we highlight (Cisco, Programming for Network Engineers, 2016):

1. Low cost: Solutions using automation reduce the complexity of network operation and the time required to provision, configure and operate network devices.

2. Business continuity: Automation reduces human errors resulting from the manual process of network configuration and operation, thus ensuring the time spent between a product idea and its launch.

3. Business agility: Repeated tasks can be automated, allowing for greater productivity.

Network automation is a solution for reducing operational costs (OPEX), improving not only the time spent configuring network devices, but also the efficiency of network maintenance through procedures that are easier to follow and implement on a large scale.

The lack of autonomy of the Spanning Tree protocol and its variants in not being able to provide traffic balancing autonomously, makes Traffic Engineering a limited and practically non-existent performance at the level of the layer two network. Therefore, the advent of software-defined networks and the algorithms of the Python programming language, allow traffic engineering to be performed autonomously in the Switched network, over the layer two protocol Per Vlan Spanning Tree + filling the gaps existing at the level of the layer two protocols.

METHODS

The preparation of this work referred us in the first instance to the empirical method, based on observation and experimentation techniques. Therefore, in its genesis, the observation technique was used through the literature review, through bibliographic analysis with specificity to traffic engineering, Ethernet switched networks, Per Vlan Spanning Tree protocol, software-defined networks and network programming.

From the analysis of the elements surrounding the research through the observation technique, the modeling technique was used, through the implementation of a virtual environment using the PNETLAB network simulator, which served as a basis for configuration, analysis and testing of the Ethernet network, using the PVST+ protocol for traffic distribution in virtual local networks.

To allow autonomy in the traffic balancing process in the switched network, implementing traffic engineering in the Ethernet switched network, the Python programming language was implemented on the devices, and the Python programming language was based on a

host with an Ubuntu operating system where algorithms were configured that allowed interaction with the devices within the virtual network environment.

The experimentation technique was materialized with the Python programming language that allowed manipulating the autonomy of the traffic balancing process.

The results obtained allowed us to measure the autonomy in the process of switching and balancing traffic in the network, leading us to optimize Traffic Engineering in the Switched Ethernet network over the Per Vlan Spanning Tree + protocol.

RESULTS

The experiments carried out to optimize traffic engineering in the Ethernet switched network were based on a network environment with specification in 2G/3G/4G RAN services, with an impact on the communication of some of the radio services over the IP protocol.

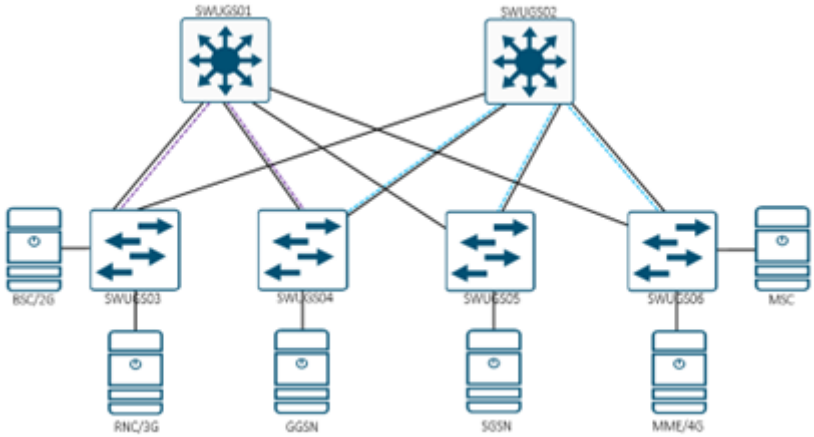

Fig. 6 Layer 2 network topology.
Source: Own authorship.

At the Ethernet switched network level, the SWUGS01 and SWUGS02 switches operate at the distribution layer, aggregating and forwarding traffic coming from the access network. In this topological structure, the distribution switches at the PVST+ protocol level have a priority (PID) equal to zero and four thousand and ninety-six respectively, thus assuming the role of root bridge. Depending on the values defined as PID, SWUGS01 assumes the role of root bridge for Vlans 21, 23, 24, 25, 27, 28, 29, 31, making the distribution switch (SWUGS01) the reference point for all Spanning Tree calculations.

The distribution switch (SWUGS02) assumes the role of primary root bridge for VLANs 32, 33, 34, 35, 100 by configuring the PID value equal to zero. Thus, for these VLANs, the traffic flow preferentially follows SWUGS02. Despite this scenario, the traffic originating from Switches SWUGS01 and SWUGS02 is forwarded to the Router Core (RT-PE01-CORE-UGS) and from there to the IP/MPLS backbone. However, at the traffic engineering level, to provide redundancy and load balancing within the Ethernet Layer 2 switched network, all VLANs existing on SWUGS01 were replicated on SWUGS02 by customizing the PID values, which allowed the priority traffic of SWUGS01 to be secondary on SWUGS02 and vice-versa.

With the network converged and the PVSTP+ protocol configuration parameters defined in accordance with the tables above, there is a traffic profile based on the routing of virtual local networks to their root bridges (SWUGS_01 and SWUGS_02). Despite this profile, the interconnection interfaces between the platforms and network devices have an installed capacity of 10 Mbps.

Traffic distribution using PVST+

With the network converged and the configuration parameters of the PVSTP+ protocol defined, there is a traffic profile based on the routing of the virtual local networks to their root bridges (SWUGS_01 and SWUGS_02). Despite this profile, the interconnection interfaces between the platforms and network devices have an installed capacity of 10 Mbps.

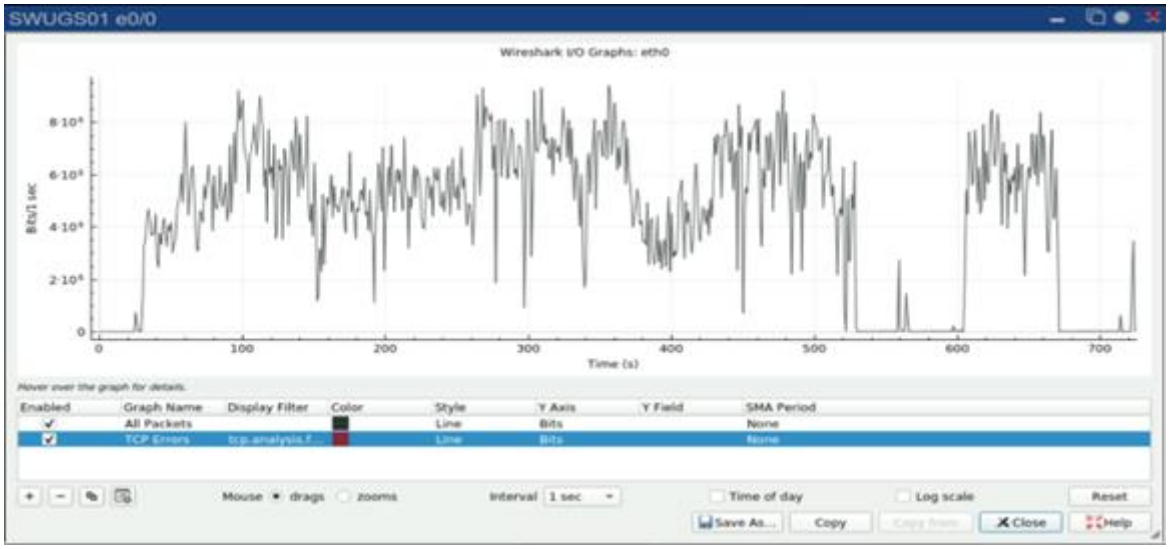

Fig. 7 -Installed capacity on the Eth0/0 interface of SWUGS_01.
Source: Own authorship.

SWUGS_01 acting as a root bridge for eight virtual private networks and SWUGS_02 acting as a root bridge for five virtual private networks, each of which reaches approximately 1 Mbps, equivalent to ten percent of its maximum capacity, below the traffic profile of the switched Ethernet network.

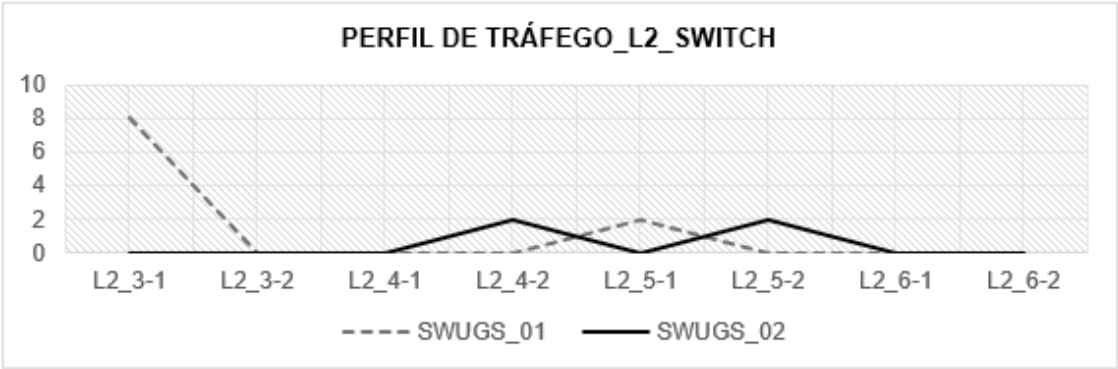

Fig. 8- Traffic profile of distribution switches.
Source: Own authorship.

PVST+ Throttling Scenario on Ethernet Network

The circuit between SWUGS_03 and SWUGS_01 has a capacity of 10Mbps, applying a rate limit of 1000 Kbps (for simulation purposes), it receives traffic that reaches up to ninety percent of its maximum capacity. On the other hand, the circuit between SWUGS_03 and SWUGS_02 has an occupancy rate of up to 3Kbps, however STP is unable to autonomously distribute traffic in a balanced manner between the two circuits.

Fig. 9 -Traffic validation in SWUGS02.
Source: Own authorship.

Fig. 10 -Excessive traffic on the circuit between SWUGS_03 and SWUGS_01.
Source: Own authorship.

One of the consequences of excessive traffic flowing through a circuit is the existence of errors and packet drops that degrade the services, affecting the quality and poor performance for the end user. In this scenario, the quality of the transmission for communication between the BTS and the BSC, that is, the ABIS radio interface, was affected by packet drops and timeouts, as a result of the excess traffic occurring on the circuit, despite having an idle redundant circuit.

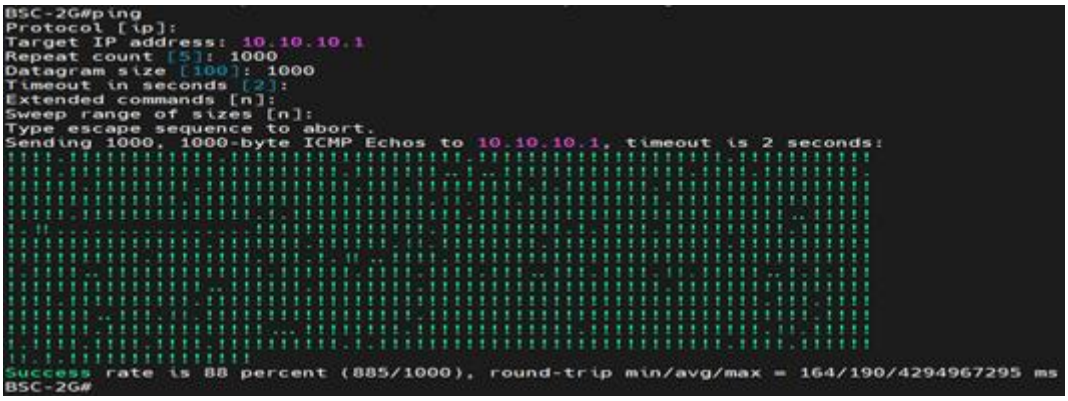

Fig. 11-Timeout in BSC due to excessive traffic between SWUGS_03 and SWUGS_01. Source: Own authorship.

Fig. 12- Drops in the circuit between SWUGS_03 and SWUGS_01. Source: Own authorship.

The scenarios presented above clearly show the limitations of STP and its variants, and the fact that it is not able to perform autonomous traffic balancing, since there is excessive traffic on one circuit and idleness on another, which shows poor end-user performance. Therefore, manual intervention is urgently needed, with the network engineer analyzing the traffic and deciding to selectively migrate part of it, ensuring the normal functioning of the services.

For small networks, human intervention is reasonable to overcome the inability of STP. However, for large and complex networks, human intervention is not scalable, requiring the implementation of an automation mechanism, which complicates the needs of the network, as well as the inability of native functions of the STP protocol within the switched network.

Automation in switching on the Ethernet Network

To carry out the automation process for switching and balancing traffic over the PVSTP+ protocol, the traditional network and the operation of the protocols that operate over it were maintained, using Python programming algorithms over the same network structure,

adding tools that were capable of instructing and executing functions outside the scope of the native operation of PVSTP+ in a particular way.

A docker with the Ubuntu operating system was necessary to accommodate the python programming language in its version 3.8.2, where it was used as a network controller, and from there the Netmiko library was imported, thus guaranteeing the SSH connection with the network devices.

Traffic switching to the switches configured and interconnected to the Python controller followed the logical sequence, as shown in the flowchart below.

The traffic profile presented in the flowchart was natively defined and forwarded according to the characteristics of the PVSTP+ protocol, that is, in accordance with the configuration of the primary and secondary root bridge in SWUGS_01 and SWUGS_02.

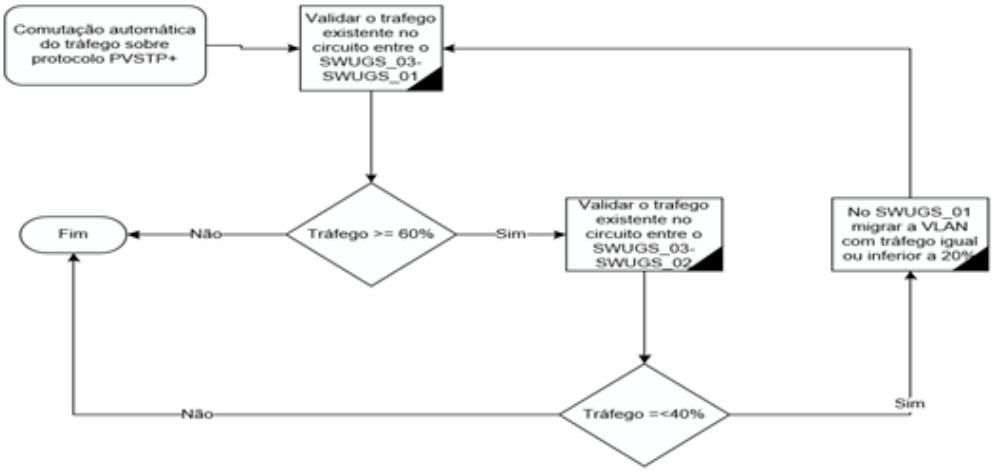

Fig. 13-Flowchart for automatic traffic switching.
Source: Own authorship.

The Abis and Iub interfaces were both configured as root bridges on SWUGS_01, with a total forwarding of 8 Mbps (4 symmetrical Mbps) on the Ethernet 0/0 interface of SWUGS_03 and 5bps on the Ethernet 0/1 interface of SWUGS_03.

Fig. 14-Abis and Iub interface traffic flowing from SWUGS03 to SWUGS01.
Source: Own authorship.

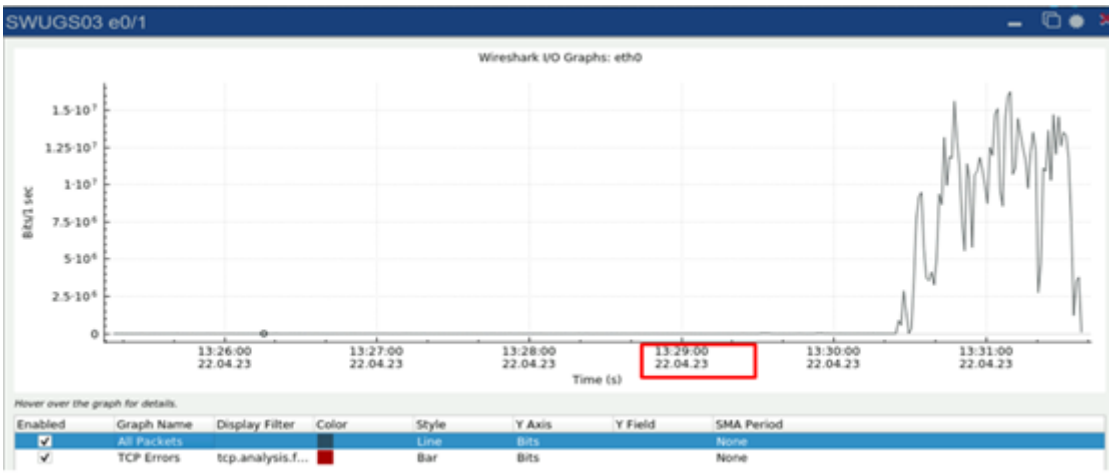

Fig. 15-Abis and Iub interface traffic flowing from SWUGS03 to SWUGS02.
Source: Own authorship.

Checking for traffic imbalance in the two root bridges, the Python programming script autonomously, that is, without human intervention, validates the traffic in SWUGS_03 so that the previously established condition is met, which is achieved by the fact that if the traffic is greater than or equal to sixty percent in the primary root bridge (SWUGS_01) and less than forty percent in the secondary root bridge (SWUGS_02), there is automatic switching.

In order to obtain more accurate traffic samples, a total traffic of 4Mbps was assumed, with the sixty and forty percent condition being reflected in traffic of 2,400,000bps and 1,400,000bps respectively.

```
output_rate_1 = net_connect.send_command("sh int ethernet 0/0 | inc rat
for x,y in enumerate(output_rate_1.splitlines()):
        if "rate" in y and "input" in y:
                val_in = y.split()[4]
        if "rate" in y and "output" in y:
                val_out = y.split()[4]


print(f"Interface Ethernet 0/0/ - IN:{val_in} OUT:{val_out}")


if (int(val_in) >= 2400000) or (int(val_out) >= 2400000):
        output_rate_2 = net_connect.send_command("sh int ethernet 0/1 |
        for b,c in enumerate(output_rate_2.splitlines()):
                if "rate" in c and "input" in c:
                        val_in = c.split()[4]
                if "rate" in c and "output" in c:
                        val_out = c.split()[4]
        print(f"Interface Ethernet 0/1 - IN:{val_in} OUT:{val_out}")

        if ( int(val_in) < 1600000 ) or ( int(val_out) < 1600000 ) :
                print("\n\nEFECTUAR A COMUTACAO DO TRAFEGO")
                subprocess.run(["python3", "pe01_core_ugs.py"])
```

Fig. 16 - Validations of automatic traffic balancing conditions.
Source: Own authorship.

Considering that the configuration of the SVIs (Switch Virtual Interface) was carried out on the core router (RT_PE01-CORE-UGS_BOX1), to allow the routing of Vlan traffic, the script accesses the core router to obtain validation of the real traffic on the interfaces, since the switching process is done for Vlans that present greater traffic and that contribute to the imbalance of the same in the network.

```
addresses_ = ["192.168.100.2"]
Username,Password = "cisco","cisco"

def backup_rtr_configuration(_IP_):
        return_net_connect = login(_IP_,Username,Password,"cisco")
        main_function(_IP_,return_net_connect,"cisco")

with concurrent.futures.ThreadPoolExecutor() as exe:
        if addresses_ != []:
                results = exe.map(backup_rtr_configuration, addresses_)
        elif addresses_ == []:
                pass
```

Fig. 17- Access to Router Core RT_PE01-CORE-UGS_BOX1.
Source: Own authorship.

```
output_rate_1 = net_connect.send_command("sh int vl21 | inc rate")
for x,y in enumerate(output_rate_1.splitlines()):
        if "rate" in y and "input" in y:
                val_in = y.split()[4]
        if "rate" in y and "output" in y:
                val_out = y.split()[4]

traf_vl21 = int(val_in + val_out)
print(f"Trafego total da interface Vlan21: {traf_vl21}bits/sec")


output_rate_2 = net_connect.send_command("sh int vl27 | inc rate")
for b,c in enumerate(output_rate_2.splitlines()):
        if "rate" in c and "input" in c:
                val_in = c.split()[4]
        if "rate" in c and "output" in c:
                val_out = c.split()[4]

traf_vl27 = int(val_in + val_out)
print(f"Trafego total da interface Vlan27: {traf_vl27}bits/sec")
```

Fig. 18- Validation of SVI traffic on Router Core RT_PE01-CORE-UGS_BOX1.
Source: Own authorship.

In the scenario created, the circuit between SWUGS_03 and SWUGS_01 presented a traffic of more than sixty percent, that is, 4Mbps and the circuit between SWUGS_03 and SWUGS_02 presented a traffic of 5bps. In this scenario, the script accessed SWUGS_01, automatically balancing the traffic by changing the root bridge of Vlan 27, increasing its priority to 8192.

```
net_connect_ = login("192.168.100.4","cisco","cisco","cisco")
host = net_connect_.send_command("sh runn | inc hostname")
print(host)

if int(traf_vl21) > int(traf_vl27) : #condicao para vlan 21
        if not net_connect_.check_enable_mode():
                net_connect_.enable()
        change = net_connect_.send_config_set(["spanning-tree vlan 21 p>
        print(change)


elif int(traf_vl21) < int(traf_vl27): #condicao para vlan 27
        if not net_connect_.check_enable_mode():
                net_connect_.enable()
        change = net_connect_.send_config_set(["spanning-tree vlan 27 p>
        print(change)
```

Fig. 19 - Automatic traffic balancing in SWUGS.
Source: Own authorship.

After executing the script, an automatic balancing of traffic between the two distribution switches was verified, since SWUGS_01 autonomously became the root bridge for Vlan 21, while SWUGS_02 became the root bridge for Vlan 27, thus balancing traffic in the network. This time, it was possible to obtain approximately 3.24 Mbps in the circuit between SWUGS_03 and SWUGS_02, for the Iub interface and approximately 2.6 Mbps in the circuit between SWUGS_03 and SWUGS_01 for the Abis interface.

Fig. 20 - Automatic traffic balancing in SWUGS.
Source: Own authorship.

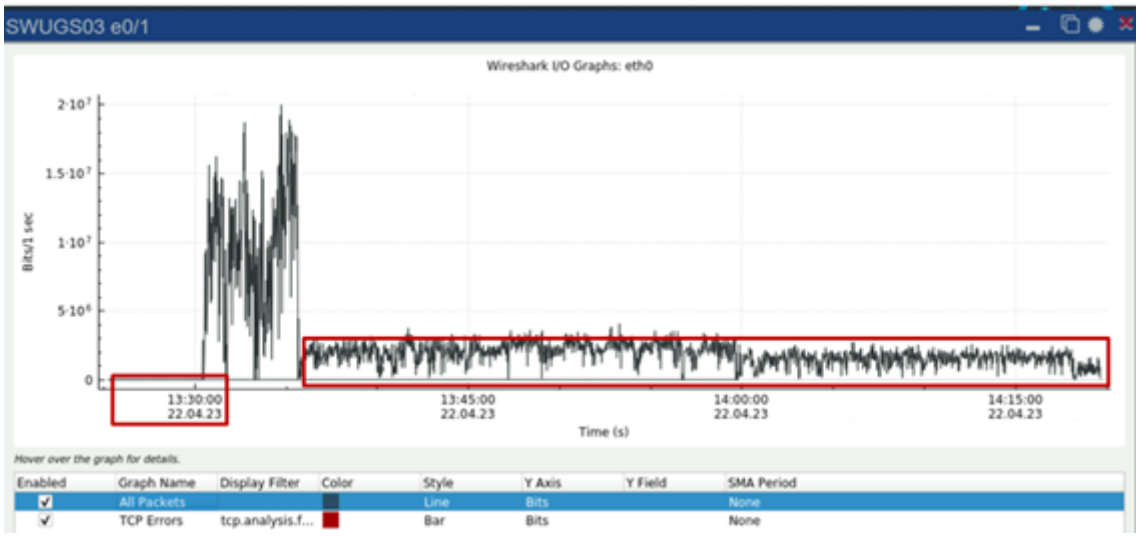

Fig. 21-Balanced traffic on the SWUGS03 Iub interface.
Source: Own authorship.

CONCLUSIONS

The work in question showed the inability of the Spanning Tree protocol and its variants, such as PVST+, to perform autonomous traffic balancing in the layer two network. It was found that natively the STP protocol and its variants are unable to perform automatic traffic switching on the various circuits existing in the switched network, even when there are congested and idle circuits. Once this scenario was validated, it was found that the STP protocol was unable to look at the size of the traffic initially existing on SWUGS01 and migrate part of it to SWUGS02, which was configured as a secondary root bridge, according to the scenario created for experimentation.

The proposal presented, which consisted of implementing a script with Python programming language, was able to satisfy the traffic engineering in the Ethernet switched network, since after its application it was possible to verify the automatic switching of traffic in the scenario where the circuit between SWUGS03 and SWUGS01 reached a traffic in Mbps equal to or greater than sixty percent of its installed capacity. In this way, the percentage values distributed in the connections to the primary and secondary root bridge reflected the traffic balancing proposed when the work began.

The flexibility of the Python programming language allowed the insertion of conditions within the programming script that automatically changed the priority values of the STP protocol from zero to four thousand and ninety-six (vice versa), these being the fundamental elements when deciding to route traffic to the redundant circuit.

The implemented scenarios made it possible to overcome the limitation of the native STP functions that were unable to automatically balance traffic, thus achieving the traffic engineering process in the Ethernet Switched network using a programming algorithm on the Per Vlan Spanning Tree protocol.

REFERENCES

Alt, B. (2018). Automation with Python. Birminghan: Packt Publishing.

Aly, B. (2018). Hands-On Enterprise Automation with Python. Mumbai: Packt Publishing.

Barroso, D., Ulinic, M., & Byers, K. (2021). NAPALM. Sphinx.

Buresh, B., Daugherty, B., Obediente, C., Roberts, E., Pfeifer, J., Garreau, K., Escalona, T. (2017). Programability and Automation with Cisco Open NX-OS. San José: Cisco.

Cisco. (2016). Programming for Network Engineers. Indianápolis: Cisco Digital Learning.

Cisco. (2020). Introduction to Network. Cisco Network Academy.

Cisco. (2020). Routing Switching and Wireless Essencials. Cisco Network Academy.

Cisco. (2020). Switching Routing and Wireless Essentials. San Frnacisco: Cisco Networking Academy.

Cisco. (2020). Switching Routing and Wireless Essentials. San Frnacisco: Cisco Networking Academy.

Cisco. (2021). Programming for Network Engineers. Cisco.

Cisco. (April de 2022). Cisco Systems, Inc. Obtido de Cisco Systems, Inc.: https://www.cisco.com/c/pt_br/support/docs/lan-switching/spanning-tree-protocol/24248-147.html

Cisco, A. (2020). Switching Routing and Wireless Essentials. San Frnacisco: Cisco Networking Academy.

Cong, H. T., Quoc, C. L., & Thuy, M. T. (2010). Study On Any Transport Over MPLS. Institute Of Technology Vietnam. Vietnam: Institute Of Technology Vietnam.

Edgeworth, B., Rios, R. G., Goley, J., & Hucaby, D. (2020). CCNP and CCIE Enterprise Core. Indianápolis: Cisco Press.

Fonseca, F. V., & da Silva, F. A. (2019). Multi-Protocol Label Switching. Obtido de https://www.gta.ufrj.br/ensino/eel879/vf/mpls/

Forouzan, B. (2010). Comunicação de Dados e Redes de Computadores. New York: AMGW Editora.

Ghein, L. D. (2016). MPLS Fundamentals. Indianápolis: Cisco Press.

Halterman, R. L. (2014). Fundamentals of Programming Python. Chicago: Southern Adventist University.

Hooda, S., Kapadia, S., & Krishnan, P. (2014). Using Trill and FabricPath and VXLAN. Indianapolis: Cisco Press.

Hucoby, D. (2005). CCNP Routing and Switching . Indianapolis: Ciscopress.

IETF. (2005). Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture. Morrisville: RFC 3985.

IETF. (16 de June de 2006). Structure-Agnostic Time Division Multiplexing over Packet. Obtido de https://datatracker.ietf.org/doc/html/rfc4553

IETF. (2006). Structure-Agnostic Time Division Multiplexing over Packet. RFC 4553.

IETF. (2007). Pseudowire Emulation Edge-to-Edge (PWE3) Asynchronous Transfer Mode (ATM) Transparent Cell Transport Service. Englewood: Rfc-4816.

Juniper. (23 de March de 2020). Juniper Networks. Obtido de Juniper Networks: https://www.juniper.net/documentation/en_US/junos/topics/concept/tdm-cesopsn-overview.html

Lammle, T. (2016). Introducing Cisco Data Center. Indianapolis: Sybex.

Lopes, J. P. (2012). Switches auto configuraveis. Lisboa: Universidade de Aveiro.

Marconi, M. d., & Lakatos, E. M. (2003). Fundamentos de Metodologia Científica. São Paulo: ATLAS S.A.

Melis, G. (2020). Network automation using Python. Internacional Hellenic University.

Melo, A. F. (2009). Engenharia de Tráfego de Redes Ethernet baseadas em Árvores de Suporte. Aveiro: Universidade de Aveiro.

Mendes, J. P. (7 de April de 2017). Arquitetura Psudowire: Uma infraestrutura multiserviços. Curitiba: Universidade Tecnológica Federal do Paraná. Obtido de https://ondemandelearning.cisco.com

NEC. (2007). Mobile Backhaul Evolution. NEC. Tóquio: NEC Corporation. https://my.nec.com/en_MY/products/carrier/Whitepaper-Mobilebackhaulevolution.pdf

Oliveira, S. A., & Mendonça, A. P. (2018). Programação para Admnistradores de Redes de Computadores. Amazonas: Instituto Federal Amazonas.

Perrin, S. (2018). A TDM to IP Solution. Heavy Reading.

Rappaport, T. S. (2009). Comunicação sem fio: Principios e práticas. São Paulo: Pearson Prentice Hall.

Rhodes, B., & Goerzen, J. (2010). Foundations of Python Network Programming. New York: Apress.

Smith, S. (2003). Introduction to MPLS. Indianápolis: Cisco.

Sverzut, J. (2008). Rede GSM. São Paulo: Érica.

Tanenbaum, A. S. (2011). Redes de computadores. Amsterdam: Campus.

Vieira, A. d. (2010). Optimização do protocolo EAPS - Ethernet Automatic Protection. Porto Alegre: UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL.

Wiki. (11 de May de 2022). wiki. Obtido de Artigos.wiki: https://artigos.wiki/blog/en/Multiple_Spanning_Tree_Protocol

Declaration of conflicts of interest:

The author of the article declares that there is no conflict of interest that affects the publication of the article.

Authorship Contribution:

The author also contributed to the conception, design and bibliographic research, which enabled the development and review of the content for final approval of the version to be published.