Sebastião Guilherme, E. (2024). Optimização da engenharia de tráfego na rede comutada ethernet usando algorítimo de programação sobre o protocolo per Vlan Spanning Tree +. Gregório Semedo: Ciencia Y Desarrollo, 1(1), e20
https://revistacdugs.ao/article/view/20

Artigo Original

REVISTA GREGÓRIO SEMEDO CIÊNCIA E DESENVOLVIMENTO

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

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

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

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

Recebido: 26 de october de 2024
Aceite: 15 de dezembro de 2024

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.

ABSTRACT
Networks currently have a large number of platforms that define communication, requiring efficient planning, as well as an adequate network design at the level of layer two and layer three using the OSI model as a reference. Therefore, at the layer three level there is traffic engineering with specificity in MPLS network traffic engineering that provides resources to manage calls and traffic depending on the network's needs. However, layer two, the Spanning Tree Protocol (STP) and its variants do not provide the same as layer three when it comes to traffic engineering. Traffic engineering in switched networks is practically non-existent due to the limitations of the STP protocol and its variants due to the impossibility of independent traffic balancing. Notwithstanding these limitations, programmable networks allow the implementation of configurations natively limited by the STP protocol, making it possible to optimize traffic engineering on the switched Ethernet network using the Python programming language.

Keywords: Traffic engineering; switched networks; 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.

INTRODUÇÃO

O crescimento rápido e contínuo das tecnologias de acesso rádio trouxe uma maior complexidade de roteamento e de conectividade global por parte dos provedores de serviço, particularmente para a telefonia móvel celular.

O advento dos serviços RAN 4G/5G, trouxe um novo paradigma de encaminhamento, onde o tráfego entre plataformas envolventes é realizado sobre uma rede de transporte orientada a comutação de pacotes usando o protocolo de internet, IP.

O problema dos provedores de serviços se remete na existência de várias redes de transporte para proporcionar o encaminhamento dos serviços 2G/3G, 4G/5G.

O protocolo MPLS vem como uma solução capaz de proporcionar uma rede de transporte convergente, servindo como *backhaul* para os mais variados tipos de tráfego, sejam eles orientados a comutação de circuitos ou comutação de pacotes.

Inevitavelmente a rede comutada aparece como elemento crucial para interligação de várias plataformas e interfaces que permitem a comunicação da camada de acesso até ao Core da rede, especificamente para rede de provedores de telefonia móvel, onde as redes locais virtuais (*VLAN*) transportam o tráfego nos mais variados destinos da rede.

Em redes de provedores de serviços, especificamente para redes móveis celulares, a integração de várias tecnologias de acesso à rede (2G/3G/4G/5G), numa mesma rede transporte, obriga que as interfaces comunicação de rádio e core possam estabelecer comunicação sobre a rede MPLS e particularmente sobre a rede comutada (NEC, 2007).

A nível da camada 3, a rede proporciona mecanismos que permitem excluir *loops* de roteamento através do campo *Time to Live* do cabeçalho IP, bem como gerir cenários de congestionamento através de engenharia de tráfego MPLS-TE.

A nível da camada dois, especificamente na rede comutada *Ethernet*, o protocolo *Spanning Tree* (STP) e as suas variantes aparecem como o único elemento capaz eliminar *loops* de camada 2, tendo em conta que a estrutura do quadro *Ethernet* não possui um campo com esta função específica (Forouzan, 2010).

Entretanto, após o estabelecimento da convergência da rede comutada, tendo uma rede livre de *loops*, o incremento de tráfego, face ao mau dimensionamento da rede, ou mesmo devido ao perfil de tráfego dos usuários que é crescente pode trazer cenários de congestionamento em determinados *links* que interligam a rede.

Per Vlan Spanning Tree

O *Per Vlan Spanning Tree* (PVSTP) opera uma instância separada de STP para cada Vlan, permitindo que o STP em cada Vlan seja configurado de forma independente, oferecendo melhor desempenho e ajuste para condições específicas (Cisco, 2020).

Várias árvores geradoras também fazem balanceamento de carga em circuitos redundantes quando os mesmos são atribuídos a diferentes Vlans. Assim, um circuito pode encaminhar um conjunto de Vlans, enquanto outro redundante pode encaminhar um conjunto diferente (Cisco A., 2020).

Os provedores de serviços, com especificidade para os serviços 2G/3G/4G/5G, actualmente operam na sua maioria sobre a rede IP/MPLS, bem como na rede comutada Ethernet, permitindo a ligação entre as diferentes interfaces de serviço, sendo que cada uma das interfaces que operam sobre a rede IP/MPLS e sobre a rede comutada *Ethernet* são segmentadas Vlans. Neste cenário, as Vlans permitem a partição de uma rede local em diferentes segmentos lógicos, permitindo que as plataformas fisicamente ligadas a diferentes switches estejam virtualmente ligados a um único *switch*. Uma vez que as Vlans

proporcionam alta flexibilidade numa rede local, estas redes tornam-se ideais em ambientes de vários tipos de tráfegos que precisam ser gerenciados (Melo, 2009).

Para obter redundância a nível de serviços, as interfaces que operam sobre a rede de camada dois possuem Vlans redundantes, configurado nos *switches,* distribuídos nos circuitos redundantes.

Se uma rede comutada Ethernet permitisse o encaminhamento de tráfego de serviço 2G/3G com uma média de 1Gbps na hora de maior movimento (HMM), a rede teria aproximadamente vinte locais virtuais e 20Gbps na infraestrutura da rede.

Num ambiente de rede de camada dois, onde é usado o PVST, os parâmetros STP são atribuídos de forma que as Vlans sejam igualmente distribuídas por todas ligações activas. Desta forma, existirão instâncias de Vlans em cada um dos links activos, conseguindo-se obter resultados óptimos em relação ao balanceamento de carga, sendo mantida uma instância STP para cada Vlan. Num cenário com vinte redes privadas virtuais afectos a redes de serviço RAN, teremos vinte instâncias para topologias lógicas diferentes, o que a princípio se considera positivo a nível de balanceamento de carga, mas com impacto negativo a nível de memória e processamento do switch uma vez que cada Vlan implica a existência de uma instância de Spanning Tree. Portanto surgiu a necessidade de diminuir o número de instância de *Spanning Tree* para mapear um conjunto de Vlan através da implementação do *Multiple Spaning Tree* (MSTP) (Edgeworth, Rios, Goley, & Hucaby, 2020).

Multiple Spanning Tree

O protocolo *Multiple Spanning Tree* (MSTP), definido pelo padrão IEEE 802.1.s apresenta-se como uma evolução do protocolo RSTP, com o objectivo de garantir a existência de várias redes locais virtuais em um número reduzido de instâncias de STP, utilizando uma rede local virtual para controle, permitindo assim uma convergência mais rápida e ainda a possibilidade de balanceamento de carga (Melo, 2009).

Cada uma das instâncias MSTP está associado a um conjunto de Vlans pertencentes a uma administração única denominada por região. Assim, uma região é uma instância independente de outra MSTP, cenário que permite realizar o balanceamento de carga das instâncias MSTP permitindo que o tráfego das redes privadas virtuais associadas as mesmas possam ser encaminhadas independentemente de outras regiões (Vieira, 2010).

A redução do número de instâncias de STP e a garantia do balanceamento de carga entre os links fazem do protocolo MST ser uma escolha a nível dos protocolos de camada dois em relação aso seus antecessores, no entanto o protocolo STP como todas variantes, não possuem inteligência suficiente para gerir de forma automática cenários de congestionamento na rede de camada dois (Edgeworth, Rios, Goley, & Hucaby, 2020).

Na perspetiva da qualidade de transmissão, mesmo que seja verificado valores de latência acima ou abaixo do *threshold*, perdas de pacotes num determinado caminho onde o protocolo PVSTP + acredita ser o melhor caminho para encaminhar o tráfego, ou qualquer outro cenário anómalo nos links entre *trunks* de transmissão, o PVSTP não fará comutação de tráfego para um *link* com melhores requisitos de transmissão de maneira autónoma, apresentando-se assim limitado no que tange a engenharia de tráfego.

Dada a flexibilidade que existe nas infraestruturas de rede e com o advento das redes definidas por *software*, os algorítimos da linguagem de programação em *Python*, permitem

realizar de maneira autónoma a engenharia de tráfego na rede Comutada, sobre o protocolo de camada dois *Per Vlan Spanning Tree +.*

Redes Definidas por Software

As redes tradicionais não são tão flexíveis quanto poderiam ser e as despesas operacionais são tão altas, isto porque muitas vezes os ambientes de rede não têm sido capazes de responder às solicitações de mudança dentro do tempo desejado (Academy, 2019).

Os provedores gostariam de fazer mudanças para acomodar as necessidades de seus clientes e, como são hoje, as redes não podem atender a essas demandas. As redes exigem uma injeção de controle baseado em software para possibilitar mudanças imediatas e oportunas para atender a essas necessidades (Learning, 2021).

Os grandes domínios de rede consistiam em um número gerenciável de routers, switches e outros dispositivos, de modo que os engenheiros de rede podiam configurar e manter esses dispositivos usando métodos manuais. Com a crescente carga na rede que é impulsionada por vídeo, voz, mídia social, integração de serviços através do protocolo MPLS (ATOM) armazenamento e aplicativos baseados em cloud, o número e a complexidade dos dispositivos de rede aumentaram a um ponto em que o gerenciamento deve ser automatizado em algum nível. A programação da rede pode fornecer capacidade de gestão eficiente dos dispositivos por meio de software (Learning, 2021).

Em uma arquitetura tradicional de *router* ou de *switch*, as funções do plano de controle e do plano de dados ocorrem no mesmo dispositivo. As decisões de roteamento e de encaminhamento de pacotes são responsabilidade do sistema operacional do dispositivo (Cisco, 2020).

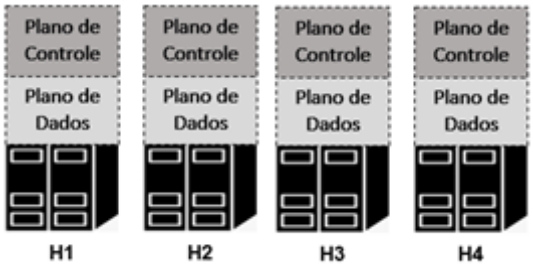

Fig. 1 - Arquitetura de Rede tradicional para encaminhamento de tráfego.

A redes definidas por software consiste basicamente na separação do plano de controle do plano de dados. Desta forma, a função do plano de controle é removida de cada dispositivo e é executada por um controlador centralizado. Este controlador centralizado, comunica as funções do plano de controle para cada dispositivo, onde cada um pode concentrar-se em no envio efectivo dos dados, enquanto o controlador centralizado gerencia o fluxo de dados, aumentando a segurança e fornecendo outros serviços (Hadley, Nicol, & Smith, 2017).

O controlador é uma entidade lógica que permite que os administradores de rede gerenciem e determinem como o plano de dados de switches e roteadores devem lidar com o tráfego da rede, orquestrando, fazendo mediações e facilitando a comunicação entre aplicativos e elementos de rede (Cisco, 2020).

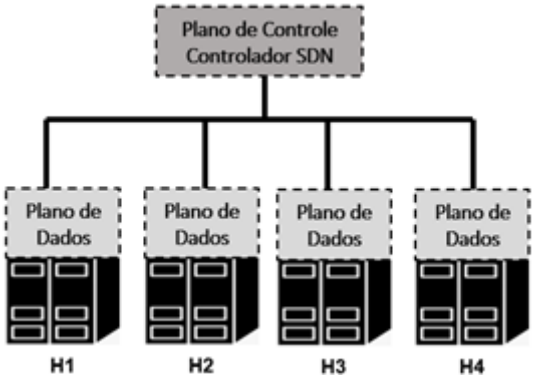

Fig. 2 - Arquitetura de Rede Definida Por Software.

As estruturas de redes definidas por software usam interfaces de programação de aplicativos (APIs), que permitem que outros aplicativos acessem seus dados ou serviços. Ela assenta-se no princípio de um conjunto de regras que descrevem como um aplicativo pode interagir com outro e as instruções para permitir que a interação ocorra (Cisco A. , 2020).

Tendo em conta as necessidades corporativas, as redes SDN podem ser baseadas em dispositivo, controlador ou ainda em políticas.

As redes definidas por software baseadas em dispositivos são programáveis por aplicativos e executados no mesmo dispositivo.

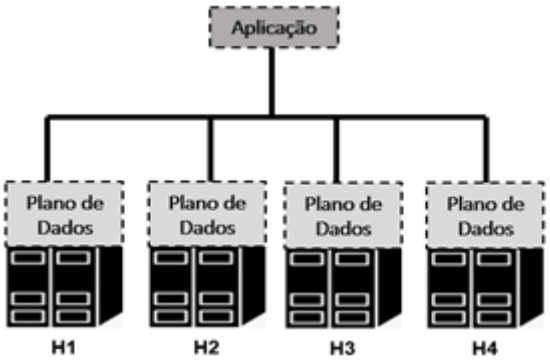

Fig. 3 - Arquitetura de Redes Definidas por Software baseada em aplicação.

As SDNs baseadas em controlador usam controladores centralizados que têm a visibilidade de todos dispositivos da rede, desta forma a aplicação consegue comunicar-se com o controlador que manipula os fluxos de dados e faz a gestão dos dispositivos.

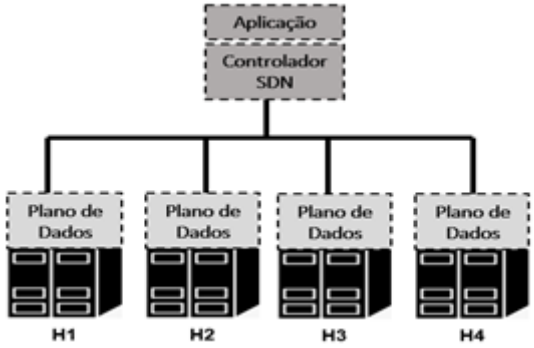

Fig. 4 - SDN baseada em controlador.

Para as SDNs basedas em políticas, um controlador central possui a visão de todos dispositivos da rede, entretanto o emento diferenciador da SDN baseda por controlador é que este usa aplicados inegrados que automatizam acções nos dispositivos atravez de uma camada com nível mais alto de abstração. A mesma possui interfaces graficas o que permite o técnico usar o mesmo sem ter habilidades de programação.

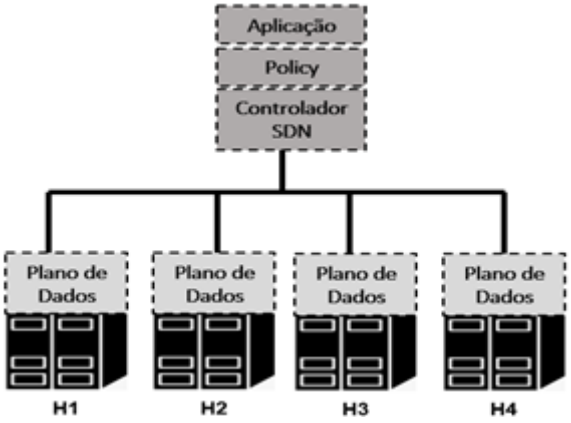

Fig. 5 - SDN baseada em políticas

Automação em Redes

A programação de rede é uma tendência aprimorada e inspirada por SDN, baseadas em métodos de *script* e linguagens de programação padrão usadas para controle e monitoramento de elementos de rede (Learning, 2021).

O conceito de SDN procura eliminar a dependência do fornecedor por meio de protocolos padrão, como o *OpenFlow*. No entanto, as redes legadas não SDN precisam manter o ritmo e responder às mudanças dinâmicas da rede. Os protocolos a nível da rede comutada como PVSTP + existem e ainda são amplamente usadas (Learning, 2021). Não obstante novos protocolos e optimizações dos mesmos, por muito mais tempo os protocolos que já vem sido considerados legados vão existir nas infraestruturas de rede.

Neste cenário aparecem alguns novos métodos de configuração de dispositivos de rede usando automação, reduzindo o tempo de configuração do equipamento e facilitando a manutenção dos mesmos. É evidente a melhoria da segurança de rede, reconhecendo e

corrigindo vulnerabilidades, aumentando a estabilidade da rede representam o futuro das redes, permitindo a gestão de um maior número de dispositivos de forma unitária (Sandu & Curpen, 2017).

Num ambiente em que existem vários dispositivos de rede e não existem várias intervenções, ou ainda existem um número reduzido de equipamentos, automação não é necessária. No entanto, o grande número de tarefas em cenários repetitivos direciona o uso de script para automatizar as funções de rede (Buresh, et al., 2017).

Além da automação das funções da rede, mais itens podem ser adicionados para que o processo de automação assuma efetivamente um papel crucial, dentre as quais destacamos (Cisco, Programming for Network Engineers, 2016):

1. Baixo custo*:* Soluções usando automação, reduz a complexidade da operação da rede e o tempo necessário para aprovisionar, configurar e operar dispositivos de rede.

2. Continuidade do negócio: A automação reduz os erros humanos fruto do processo manual de configuração e operação de rede, garantindo assim o tempo gasto entre as ideias de um produto e o seu lançamento.

3. Agilidade no negócio: Tarefas repetidas pode ser automatizada permitindo maior produtividade.

A automação da rede é uma solução para redução de custos operacionais (OPEX), melhorando não apenas o tempo gasto para configurar os dispositivos de rede, mas também a eficiência da manutenção da rede por meio de procedimentos mais fáceis de seguir e implementar em grande escala.

A falta de autonomia do protocolo Spanning Tree e suas variantes em não conseguir proporcionar balanceamento do tráfego de maneira autónoma, faz com que a Engenharia de tráfego seja uma realizada limitada e praticamente inexistente a nível da rede de camada dois. Assim sendo, o advento das redes definidas por *software* e os algorítimos da linguagem de programação em *Python*, permitem realizar de maneira autónoma a engenharia de tráfego na rede Comutada, sobre o protocolo de camada dois *Per Vlan Spanning Tree* + colmatando as lacunas existentes a nível dos protocolos de camada dois.

MÉTODOS

A elaboração deste trabalho remeteu-nos em primeira instância ao método empírico, baseando-se nas técnicas de observação e experimentação. Assim sendo, na sua génese usou-se a técnica de observação através da revisão da literatura, por meio da análise bibliográfica com especificidade a engenharia de tráfego, redes comutadas Ethernet, protocolo *Per Vlan Spanning Tree*, Redes definas por *software* e programação de redes.

Da análise dos elementos envolventes da pesquisa através da técnica de observação foi usada a técnica de modelação, através da implementação de um ambiente virtual usando o simulador de redes PNETLAB, que serviu de base para configuração, análise e testes da rede Ethernet, usando protocolo PVST+ para distribuição do tráfego nas redes locais virtuais.

Para permitir a autonomia no processo de balanceamento de tráfego na rede comutada, efetivando a engenharia de tráfego na rede comutada Ethernet, foi implementada a linguagem de programação em *Python* sobre os dispositivos, sendo que a linguagem de programação em *Python* foi assente em um *host* com sistema operacional *Ubuntu* onde sobre o mesmo foram configurados algorítimos que permitiram a interação com os dispositivos dentro do ambiente virtual de rede.

A técnica de experimentação materializou-se com a linguagem de programação em *Python* que permitiu manipular a autonomia do processo de balanceamento do tráfego.

Os resultados obtidos permitiram medir a autonomia no processo de comutação e balanceamento do tráfego na rede, conduzindo-nos a optimização da Engenharia de tráfego na rede Comutada Ethernet sobre o protocolo *Per Vlan Spanning Tree* +.

RESULTADOS

Os experimentos realizados para a optimização da engenharia de tráfego na rede comutada Ethernet, foram baseados num ambiente de rede com especificação nos serviços RAN 2G/3G/4G, com incidência na comunicação de alguns dos serviços de rádio sobre o protocolo IP.

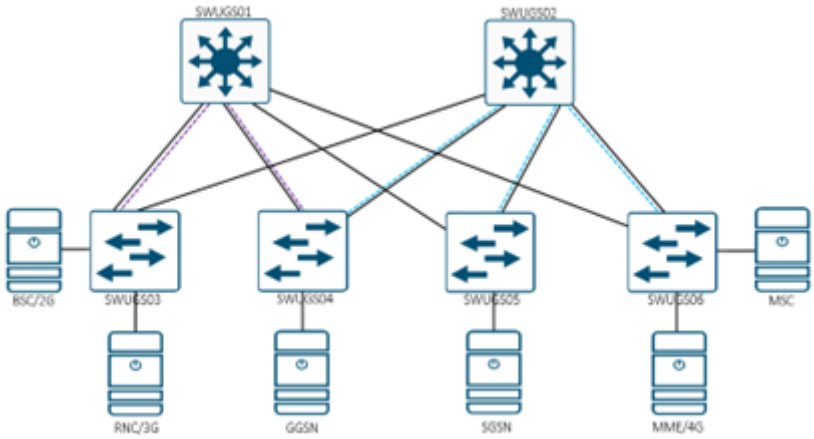

Fig. 6 Topologia de rede Layer 2.
Fonte: Autoria própria.

A nível da rede comutada Ethernet, os switches SWUGS01 e SWUGS02, operam na camada de distribuição, agregando e encaminhando o tráfego vindo da rede de acesso. Nesta estrutura topológica, os switches de distribuição a nível do protocolo PVST+ possuem uma prioridade (PID) igual a zero e quatro mil e noventa e seis respectivamente, assumindo desta forma o papel de *root bridge*. Em função dos valores definidos como PID, o SWUGS01 assume o papel de *root bridge* para as Vlans 21, 23, 24, 25, 27, 28, 29, 31, fazendo com que o switch de distribuição (SWUGS01) seja o ponto de referência para todos cálculos de *Spanning Tree*.

O switch de distribuição (SWUGS02), assume o papel de *root bridge* primário para as Vlans 32, 33, 34, 35, 100 através da configuração do valor de PID igual a zero. Assim, para estas Vlans o fluxo do tráfego segue preferencialmente o SWUGS02. Não obstante a este cenário, o tráfego oriundo dos Switches SWUGS01 e SWUGS02 são encaminhados para o

Router Core (RT-PE01-CORE-UGS) e deste para o backbone IP/MPLS. No entanto a nível de engenharia de tráfego, para prover redundância e balanceamento de carga dentro da rede comutada Ethernet Layer 2, todas Vlans existentes no SWUGS01 foram replicadas no SWUGS02 customizando os valores de PID, o que permitiu que o tráfego prioritário do SWUGS01 fosse secundário no SWUGS02 e vice-versa.

Estando a rede convergente e com a definição dos parâmetros de configuração do protocolo PVSTP + conforme as tabelas acima, existe um perfil de tráfego em função do encaminhamento das redes locais virtuais para os seus *roots bridges* (SWUGS_01 e SWUGS_02). Não obstante a este perfil, as interfaces de interligação entre as plataformas e dispositivos de rede possuem uma capacidade instalada de 10 Mbps.

Distribuição do tráfego usando PVST+

Estando a rede convergente e com a definição dos parámetros de configuração do protocolo PVSTP +, existe um perfil de tráfego em função do encaminhamento das redes locais virtuais para os seus *roots bridges* (SWUGS_01 e SWUGS_02). Não obstante a este perfil, as interfaces de interligação entre as plataformas e dispositivos de rede possuem uma capacidade instalada de 10 Mbps.

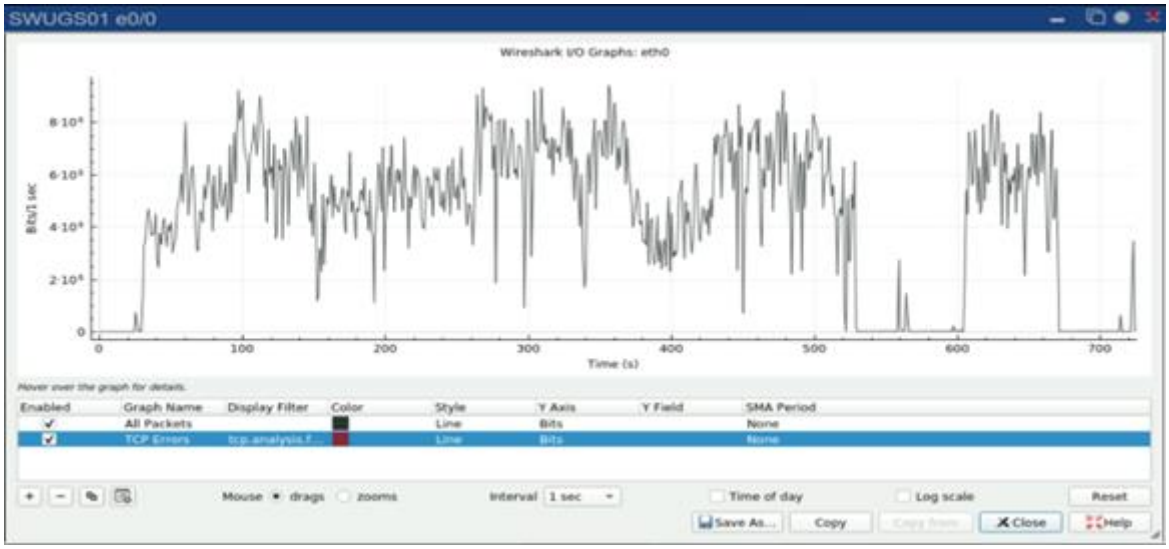

Fig. 7 - Capacidade instalada na interface Eth0/0 do SWUGS_01.
Fonte: Autoria própria.

O SWUGS_01 apresentando-se como *root bridge* para oito redes privadas virtuais e o SWUGS_02 como *root bridge* para cinco redes privadas virtuais onde cada uma delas atinge aproximadamente 1 Mbps equivalendo a dez por cento da sua capacidade máxima, abaixo o perfil de tráfego da rede comutada Ethernet.

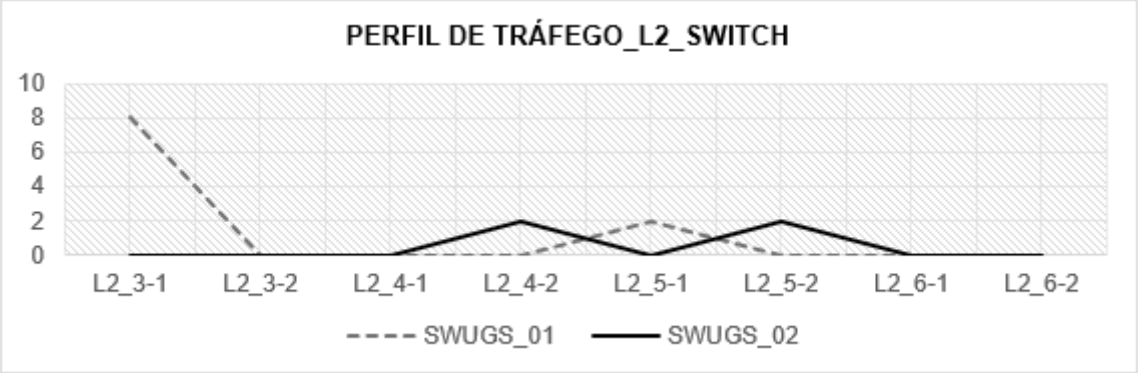

Fig. 8 Perfil de tráfego dos Switches de distribuição.
Fonte: Autoria própria.

Cenário de limitação do PVST+ na Rede Ethernet

O circuito entre o SWUGS_03 e SWUGS_01 possui uma capacidade de 10Mbps, aplicando de um *rate limit* de 1000 Kbps (para efeitos de simulação), o mesmo recebe um tráfego que atinge até noventa por cento da sua capacidade máxima. Por outro lado, o circuito entre o SWUGS_03 e SWUGS_02 encontra-se com uma taxa de ocupação de até 3Kbps, no entanto o STP não consegue de maneira autónoma distribuir o trafego de maneira balanceada entre os dois circuitos.

Fig. 9 - Validação de tráfego no SWUGS02.
Fonte: Autoria própria.

Fig. 10 - Excesso de tráfego no circuito entre o SWUGS_03 e SWUGS_01.
Fonte: Autoria própria.

Uma das consequências do excesso de tráfego que flui por um circuito é a existencia de erros e *drops* de pacotes que degradam os serviços afectando a qualidade e má performance o usuário final. Neste cenário a qualidade da transmissão para a comunicação entre a BTS para a BSC, isto é a interface de rádio *Abis,* foi afectado por *drops* de pacotes e *timeouts*, fruto do excesso de tráfego que se verifica no circuito, não obstamte termos um circuito redundante ocioso.

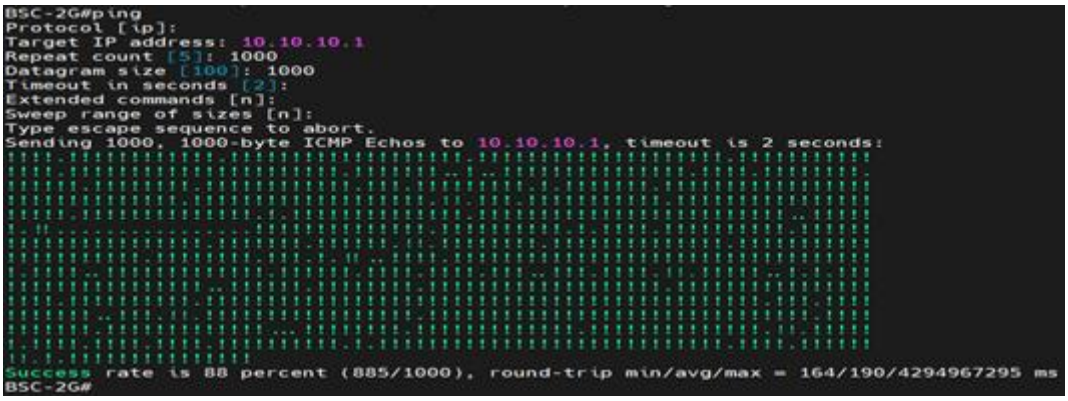

Fig. 11- Timeout na BSC face ao excesso de trafego entre o SWUGS_03 e SWUGS_01.
Fonte: Autoria própria.

Fig. 12- Drops no circuito entre o SWUGS_03 e SWUGS_01.
Fonte: Autoria própria.

Os cenários apresentados acima traduziram a clara limitação do STP e as suas variantes e em não conseguir efectuar um balanceamento autonomo do trafego uma vez que verificado excesso de trafego em um circuito e ociosidade em outro circuito, evidenciando a má performance do usuário final. Desta forma urge a necessidade de uma intervenção manual é , fazendo com o engenheiro de redes analise o tráfego e decida migrar selectivamente parte dela garantindo o normal funcionamento dos serviços.

Para redes pequenas, a acção humana apresenta-se razoável para colmatar incapacidade do STP. entretanto para redes grandes e complexas a intervenção humana não é escalável, fazendo com que fosse implementado mecanismo de automatização, colamatando as necessidades da rede, bem como a incapacidade de funções nativas do protocolo STP dentro da rede comutada.

Automatização na comutação na Rede Ethernet

Para efectuar o processo de automatização pna comutação e balanceamento do tráfego sobre o protocolo PVSTP+, mantevesse a rede tradicional e o funcionamento dos protocolos que operam sobre a mesma, sendo usado sobre a mesma estrutura de rede os algorítimos de programação em *Python* adicionando ferramentas que foram capazes de instruir e executar funções fora do escopo do funcionamento nativo do PVSTP + de maneira particular.

Um docker com a sistema operacional Ubuntu foi becessário para acomodar a linguagem de programação python na sua versão 3.8.2, onde este foi usado como controlador da rede, sendo que a partir do mesmo foi importado a biblioteca *Netmiko* garantundo assim a conexão SSH com os dispositivos de rede.

A comutação do tráfego para os switches configurados e interligados ao controlador *Python* obedeceram a sequência lógica, conforme o fluxograma abaixo.

O perfil de tráfego apresentado no fluxograma fou nativamente definido e encaminhado em função das características do protocolo PVSTP +, isto, em concordância com a configuração do *root bridge* primário e secundário no SWUGS_01 e SWUGS_02.

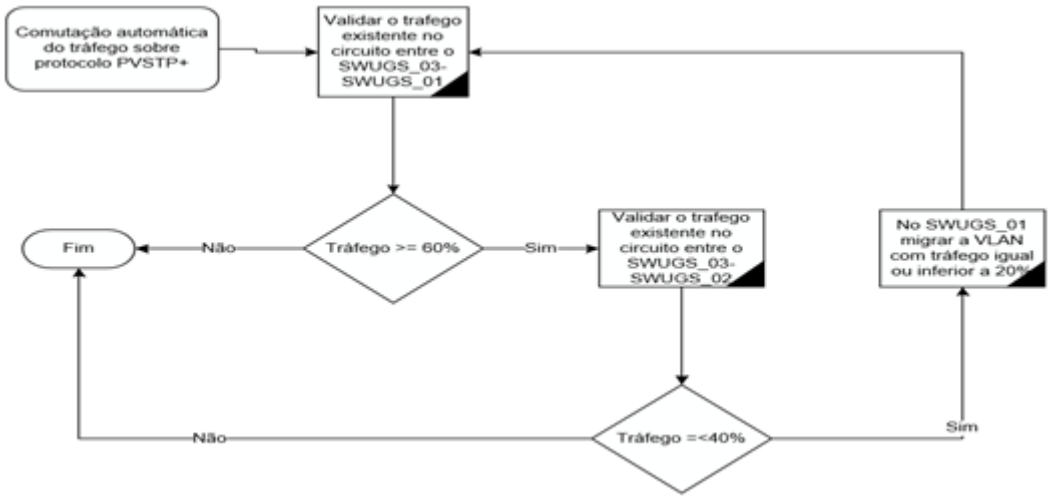

Fig. 13- Fluxograma para comutação automática do tráfego.
Fonte: Autoria própria.

A interfafce *Abis* e *Iub*, foram ambos configurados como *root bridge no SWUGS_01*, existindo um encaminhamento total de 8 Mbps (4Mbps simétricos) na interface Ethernet 0/0 do SWUGS_03 e 5bps na interface Ethernet 0/1 do SWUGS_03.

Fig. 14- Tráfego da interface *Abis* e *Iub* fluindo do SWUGS03 para o SWUGS01.
Fonte: Autoria própria.

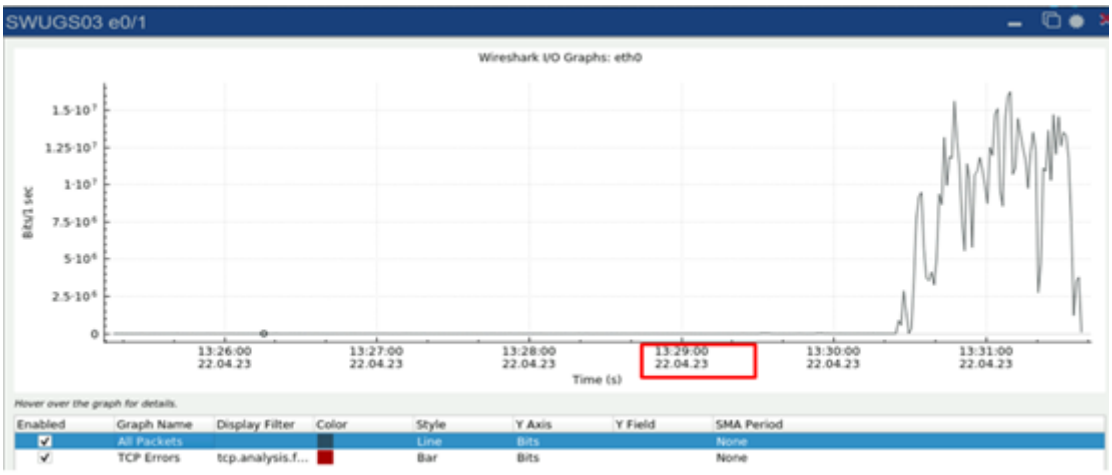

Fig. 15-Tráfego da interface *Abis* e *Iub* fluindo do SWUGS03 para o SWUGS02.
Fonte: Autoria própria.

Verificando desbalanceamento de tráfego nos dois *roots bridge*, o *script* de programação em *Python* de maneira autonoma, ou seja, sem intervenção humana, valida o tráfego no SWUGS_03 de forma que realizada a condição previamente estabelecida que se consuma no facto de que se o tráfego for superior ou igual a sessenta por cento no *root bridge* primário (SWUGS_01) e menor que quarenta por cento no *root bridge* secundário (SWUGS_02), haja comutação automática.

De maneira a obter amostras mais precisas anível de tráfego, assumiu-se um tráfego total de 4Mbps, sendo que a condição de sessenta e quarenta por cento foi reflectido no tráfego de 2400000bps e 1400000bps respectivamente.

```
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 - Validações das condições de balanceamento automático do tráfego.
Fonte: Autoria própria.

Tendo em conta que a configuração das SVIs (*Switch Virtual Interface*) fora realizada no router core (RT_PE01-CORE-UGS_BOX1), para permitir o roteamento do tráfego das Vlans, o *script* acede o router core para obter a validação do tráfego real nas interfaces, uma vez que o processo de comutação é feito para Vlans que apresentam maior tráfego e que concorrem para o desbalanceamento do mesmo na rede.

```
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 Acesso ao Router Core RT_PE01-CORE-UGS_BOX1.
Fonte: Autoria própria.

```
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- Validação do tráfego das SVIs no Router Core RT_PE01-CORE-UGS_BOX1.
Fonte: Autoria própria.

No cenário criado o circuito entre SWUGS_03 e SWUGS_01, apresentou um tráfego superior a sessenta por cento, isto é, 4Mbps e o circuito entre o SWUGS_03 e SWUGS_02 apresentou um tráfego de 5bps. Neste cenáro *sript* acedeu o SWUGS_01 efectando o balanceamento automático do tráfego atrves da alteração do root bridge da Vlan 27, aumentado a sua prioridade para 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 - Balanceamento automático do tráfego no SWUGS.
Fonte: Autoria própria.

Após a execução do script verificou-se um balanceamento automático do tráfego entre os dois switches de distribuição, uma vez que de maneira autonoma o SWUGS_01 passou a ser *root bridge* para a Vlan 21, enquanto o SWUGS_02 passou a ser o root bridge para a Vlan 27, equilibrando assim o trafego na rede. Desta feita consegiu-se obter aproximadamente 3.24 Mbps no circuito entre o SWUGS_03 e SWUGS_02, para interface Iub e aproximadamente 2.6 Mbps no circuito entre o SWUGS_03 e SWUGS_01 para interface Abis.

Fig. 20 - Balanceamento automático do tráfego no SWUGS.
Fonte: Autoria própria.

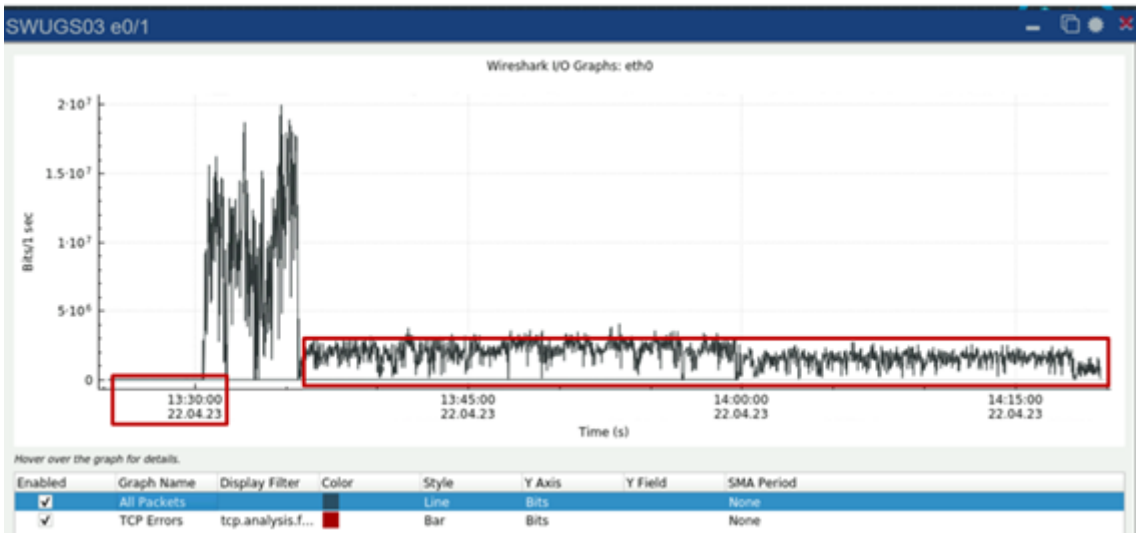

Fig. 21- Tráfego balanceado na interface Iub do SWUGS03.
Fonte: Autoria própria.

CONCLUSÕES

O trabalho em causa apresentou a incapacidade do protocolo Spanning Tree e as suas variantes como é o caso do PVST+, realizar o balanceamento autónomo do trafego na rede de camada dois. Verificou-se que nativamente o protocolo STP e suas variantes não conseguem efectuar a comutação automática do tráfego nos mais variados circuitos existentes na rede comutada, mesmo existindo circuitos congestionados e outros ociosos. Uma vez validado este cenário constatou-se a impossibilidade do protocolo STP olhar para a dimensão do tráfego existente inicialmente do SWUGS01 e migrar parte dele no SWUGS02 que foi configurado como *root bridge* secundário, conforme o cenário criado para experimentação.

A proposta apresentada que consistiu na implementação de *script* com linguangem de programção em *Python*, conseguiu satisfazer a engenharia de tráfego na rede comutada *Ethernet*, uma vez que após a sua aplicação consegui-se constatar a comutação automática do tráfego no cenário onde o circuito entre o SWUGS03 e SWUGS01 atingiu um tráfego em Mbps igual ou superior a sessenta por cento da sua capacidade instalada. Desta forma, os valores percentuais distribuídos nas ligações para o *root bridge* primário e secundário refletiu o balanceamento de tráfego proposto aquando do início do trabalho.

A flexibilidade da linguagem de programação em *Python* permitiu a inserção de condições dentro do *script* de programação que de maneira automática alteraram os valores de prioridade do protocolo STP de zero para quatro mil e noventa seis *(*vice-versa*)*, sendo estes os elementos fundamentais aquando da decisão de encaminhar o tráfego para o circuito redundante.

Os cenários implementados permitiram colmatar a limitação das funções nativas do STP que não conseguia proceder com o balanceamento automático do tráfego, conseguindo desta feita alcançar o processo de engenharia de tráfego na rede Comutada *Ethernet* usando algorítimo de programação sobre o protocolo Per Vlan Spanning Tree.

REFERÊNCIAS BIBLIOGRÁFICAS

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

Declaração de conflictos de interesses:

A autor do artigo declara não existir qualquer conflicto de interesses que afecte a publicação do artigo.

Contribuição de Autoria:

A autor contribuio igualmente na conceção, delineamento e pesquisa bibliográfica, que possibilitou o desenvolvimento e revisão do conteúdo para aprovação final da versão a publicar.