UNIVERSIDADE FEDERAL DO PARANÁ

CONRADO JOSÉ GUSSO BOZZA

MECANISMO DE DESCARTE DE PACOTE EM REDE DEFINIDA POR SOFTWARE COM SOBRECARGA NO CONTROLADOR

CURITIBA
2018

CONRADO JOSÉ GUSSO BOZZA

MECANISMO DE DESCARTE DE PACOTE EM REDE DEFINIDA POR SOFTWARE COM SOBRECARGA NO CONTROLADOR

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Engenharia elétrica, no Programa de Pós-Graduação em Engenharia Elétrica, do Setor de Tecnologia da Universidade Federal do Paraná.

Orientador: Prof. Dr. Eduardo Parente Ribeiro

CURITIBA
2018

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

B793m Bozza , Conrado José Gusso
Mecanismo de descarte de pacote em rede definida por software com sobrecarga no controlador / Conrado José Gusso Bozza . – Curitiba, 2018.

Dissertação - Universidade Federal do Paraná, Setor de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, 2018.

Orientador: Eduardo Parente Ribeiro . -

1. Interfaces (Computadores). 2. Controlador (informática). 3. Restauração. I. Universidade Federal do Paraná. II. Ribeiro , Eduardo Parente. III. Título.

CDD: 621.3981

Bibliotecária: Vanusa Maciel - CRB - 9/1928

MINISTÉRIO DA EDUCAÇÃO
SETOR TECNOLOGIA
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO ENGENHARIA ELÉTRICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em ENGENHARIA ELÉTRICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **CONRADO JOSÉ GUSSO BOZZA** intitulada: **MECANISMO DE DESCARTE DE PACOTE EM REDE DEFINIDA POR SOFTWARE COM SOBRECARGA NO CONTROLADOR**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 31 de Julho de 2018.

EDUARDO PARENTE RIBEIRO
Presidente da Banca Examinadora (UFPR)

LUIS HENRIQUE ASSUMPÇÃO LOLIS
Avaliador Interno (UFPR)

LUIS CARLOS ERPEN DE BONA
Avaliador Externo (UFPR)

Dedico este trabalho a minha família, que é berço de apoio e incentivo, e a todos que buscam o desenvolvimento científico.

AGRADECIMENTOS

Agradeço a minha família por me incentivar a sempre crescer e a não desistir, principalmente a minha mãe Paula e a minha tia Regina. A todo apoio recebido de meu orientador, prof. Dr. Eduardo P. Ribeiro, com nossas reuniões, respostas e revisões. Além do Adriano Favaro que me tirou várias dúvidas técnicas. A meus colegas da GVT que entenderam minhas ausências e incentivaram meu desenvolvimento, assim como aos amigos, e aos professores das disciplinas que me ajudaram a abrir a mente para outras perspectivas.

Também agradeço a todos que, de alguma forma, conseguiram contribuir para o desenvolvimento desta pesquisa.

“Uma caminhada de mil quilômetros começa com o primeiro passo.”
– Provérbio Chinês.

“Ninguém ignora tudo, ninguém sabe tudo. Por isso, aprenderemos sempre.”
– Paulo Freire.

RESUMO

Redes controladas por software estão ganhando espaço devido a suas vantagens, como flexibilidade e escalabilidade. Esta pesquisa se baseia em simulações de comportamentos que potencialmente podem apresentar essas redes quando submetidas à sobrecarga, situação que pode causar perda de qualidade do serviço de transmissão de dados. Neste trabalho foi testada uma solução de descarte de pacotes. Foram explorados os problemas decorrentes de congestionamento na interface entre switch e controlador para diferentes limites de capacidade de transferência. O tráfego nessa interface ocorre para instanciação de novos fluxos ou outros eventos. Verificou-se a mudança na ordenação dos pacotes, o volume de perda de pacotes e a banda correspondente nesses cenários. Esta análise permitiu estimar que o congestionamento na comunicação entre controlador e switch pode levar a degradação de desempenho a usuários e indicou que é justificável tomar medidas mitigadoras para prevenir uma degradação acentuada. Então, foi verificado o desempenho de mecanismo de descarte de pacotes tipo Blackhole como solução por comparação com o comportamento padrão do switch SDN, e ele mostrou-se melhor para o cenário avaliado.

Palavras-chave: SDN. Controlador. Interface. Restauração. Blackhole.

ABSTRACT

Software defined networks are gaining ground because of their advantages, such as flexibility and scalability. This thesis presents a research based on simulations of behaviours that networks can potentially present when subjected to overload. This situation can cause loss of quality of the data transmission service. In this work, a packet discard solution was tested. The problems of traffic increase in the interface between switch and controller for different limits of transfer capacity were investigated. The traffic on this interface occurs for new flows instantiation or others events. Packages reordering and bandwidth variation in those scenarios were verified. This analysis allowed to estimate that the congestion in the communication between controller and switch can lead to the degradation of performance for users and indicates that it justifies taking action to prevent severe degradation. The performance of a packet discard mechanism like Blackhole was verified as a solution by comparison with the standard behaviour of the SDN switch, and it proved be better for the evaluated scenario.

Keywords: SDN. Controller. Interface. Restauration. Blackhole.

FIGURAS

TABELAS

SIGLAS

BH	Blackhole
CSV	Comma-Separated Values
FFS	Fast Flow Setup
HT	Hard Timeout
IP	Internet Protocol
ONP	Open Network Foundation
OSPF	Open Shortest Path First
OVS	Open vSwitch
RMON	Remote Network Monitoring
RTP	Real-time Transport Protocol
S-C	Interface entre switch e controlador SDN
SDN	Software Define Network (Rede definida por software)
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
T-S	Interface entre Terminal de usuário e switch
UDP	User Datagram Protocol
VoD	Video On Demand

SUMÁRIO

1 INTRODUÇÃO

O desempenho no estabelecimento de novas rotas de comunicação é cada vez mais importante devido aos novos serviços sob demanda. Serviços como vídeo sob demanda (*Video On Demand* – VoD) e jogos em rede se tornaram populares, e o desempenho da estrutura de rede é essencial para garantir a experiência do usuário. Nesses serviços é sempre esperado o início imediato da transmissão e com a melhor qualidade possível. Para início rápido da transmissão, a configuração da rota dos pacotes também deve ser rápida.

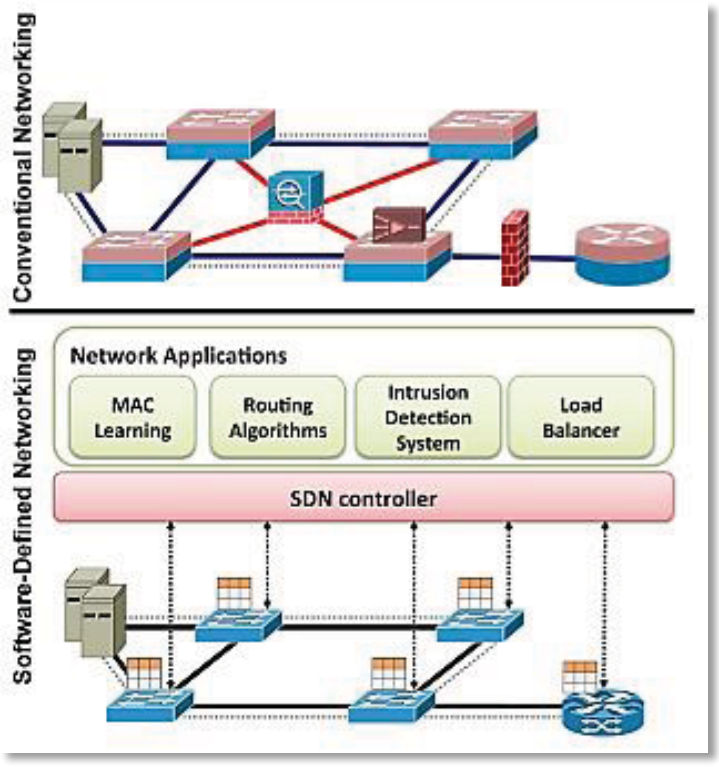
Atualmente os provedores de serviços de redes têm utilizado a arquitetura de redes definidas por software (*Software Define Network* – SDN) devido a seus diferenciais, como a flexibilidade de configuração e de gerenciamento, inclusive reduzindo custos (SCHENKER, 2011). Essas características são importantes em ambientes de alta disponibilidade de redes complexas, que normalmente contam com um centro de operação. Nessas redes, o processo de configuração de rotas é diferente das redes tradicionais.

Em SDN ocorre o desacoplamento do plano de dados e o plano de controle, em comparação à arquitetura de redes tradicionais, conforme representado na Figura 1. A centralização do controle permite maior flexibilidade e maior eficiência no gerenciamento da rede, com maior grau de abstração e com melhor visibilidade dos fluxos transportados. Em uma rede SDN as configurações são implementadas em um único ponto para todos os equipamentos, no controlador.

A rede tradicional tem a vantagem de que os elementos vão funcionar mesmo se um deles não estiver configurado e a desvantagem de ter de configurar cada um deles. Na rede tradicional cada equipamento é independente e deve ser configurado diretamente, exigindo um esforço maior para efetivar alterações. Há algumas formas de facilitar o controle de redes tradicionais com protocolos de gerenciamento, como *Simple Network Management Protocol* (SNMP) e *Remote Network Monitoring* (RMON) (Pavlou, 2007). Elas exigem o desenvolvimento de aplicativos e isso já faz parte do modelo SDN. Além de que em grandes redes o volume de equipamentos especializados (ou *Middlebox*, como *Firewalls*, coletores estatísticos, gerenciadores e outros) está semelhante

ao volume de roteadores (SHERRY, 2012). Em SDN esses equipamentos são desnecessários, pois suas funções podem ser implementadas de forma virtual.

Figura 1 – Comparativo das arquiteturas de rede tradicional e rede SDN para os mesmos serviços



Fonte: KREUTZ, 2015.

O princípio do SDN é um equipamento ativo da rede (assim como um switch) cujas funções são programadas por um controlador externo, tendo a flexibilidade de alterar comandos e parâmetros sem depender do fabricante do equipamento. Esse controlador funciona como o sistema operacional da rede, oferecendo uma visão unificada daquela, e nele podem ser implementadas as funcionalidades especificas para a solução pretendida (GUEDES, 2012). Como exemplo, pode ser configurada solução de roteamento para que os computadores ligados à rede tenham a percepção da existência de apenas um roteador.

Serviços sob demanda utilizam a internet para transferência de grande quantidade de dados. A velocidade de acesso média dos usuários brasileiros atualmente é de 6,4 Mbps, conforme divulgado pela Akamai em 2016 (AKAMAI, 2016). Existe previsão de que o tráfego na internet dedicada a negócios dobre em três anos, segundo a Cisco (2016), conforme detalhado na

Tabela 1. Nesse ambiente de crescimento, as redes SDN ganharam destaque devido a suas aplicações em servidores virtuais e vêm ganhando espaço em outras soluções. Sinais deste destaque são a realização de vários eventos e as publicações de artigos sobre SDN, nos últimos anos.

Tabela 1 – Previsão de crescimento de tráfego

IP Traffic, 2015–2020						
	2015	2016	2017	2018	2019	2020
By Type (Petabytes [PB] per Month)						
Fixed Internet	49,494	60,160	73,300	89,012	108,102	130,758
Managed IP	19,342	22,378	25,303	28,155	30,750	33,052
Mobile data	3,685	6,180	9,931	14,934	21,708	30,564
By Segment (PB per Month)						
Consumer	58,539	72,320	89,306	109,371	133,521	162,209
Business	13,982	16,399	19,227	22,729	27,040	32,165
By Geography (PB per Month)						
Asia Pacific	24,827	30,147	36,957	45,357	55,523	67,850
North America	24,759	30,317	36,526	43,482	50,838	59,088
Western Europe	11,299	13,631	16,408	19,535	23,536	27,960
Central and Eastern Europe	5,205	6,434	8,116	10,298	13,375	17,020
Latin America	4,500	5,491	6,705	8,050	9,625	11,591
Middle East and Africa	1,930	2,698	3,822	5,380	7,663	10,865
Total (PB per Month)						
Total IP traffic	72,521	88,719	108,533	132,101	160,561	194,374

Fonte: CISCO, 2016.

Apesar das vantagens, o SDN também enfrenta novos desafios, que não existem com as redes tradicionais. Uma característica é justamente a dependência de um controlador externo e da comunicação com ele. Condição que influencia o desempenho do estabelecimento de rotas, pois, a cada evento desses, o switch normalmente precisa solicitar informações do controlador.

Na rede tradicional, uma nova rota é definida manualmente ou por um protocolo de roteamento que dá autonomia para o equipamento. Na SDN, pode ocorrer a definição por demanda, e o controlador faria o papel do protocolo de roteamento. O fluxo para atualização da tabela de roteamento é maior e tem maiores riscos de levar mais tempo ou de não acontecer.

Novas rotas são instaladas a partir de novas demandas de comunicação, geralmente quando novos dispositivos são conectados à rede ou quando uma rota que não era usada há muito tempo é reativada. Rotas que

deixam de ser utilizadas são removidas para economia de recursos. Também podem ocorrer ataques maliciosos que simulam novas conexões para sobrecarregar algum dispositivo ou toda a rede.

Nesse cenário, este trabalho dá atenção especial ao desempenho da comunicação entre switch e controlador, que é essencial para instalação das rotas. Essa interface entre switch e controlador é por onde trafegam as mensagens de controle. Foi avaliado, em situações de congestionamento nessa interface, o volume de pacotes de controle não recebidos pelo switch como impacto direto e o volume de dados não recebidos pelo destino da comunicação como impacto final.

Para confirmar os impactos e permitir projetá-los para grandes escalas, foi necessário realizar testes com diferentes cenários de topologia de rede. Os testes foram em ambiente emulado, pela facilidade de se testar várias topologias, e foi selecionada ferramenta que permite utilizar equipamentos reais e simulados simultaneamente. Testes em grande escala teriam grande custo com equipamentos físicos.

Mensurados os impactos, a busca foi por mitigá-los. Eram conhecidas algumas pesquisas que apresentavam resultados que poderiam ser utilizados como solução. Então, foi avaliado o impacto de utilização da solução mais direcionada ao problema de perda de pacotes de controle por congestionamento. O Blackhole (BH) é um mecanismo de descarte de pacotes proposto para reduzir o tráfego entre controlador e equipamentos (FAVARO, 2015).

Nesta estratégia, um novo fluxo recebido pelo switch gera apenas um pacote para o controlador, pois ocorre o descarte dos pacotes do fluxo até que o controlador informe a ação e a nova regra seja adicionada na tabela do switch. O presente trabalho procura confirmar o impacto potencial dessas situações de congestionamento e avaliar a efetividade desta técnica de descarte na mitigação de seus efeitos.

1.1 OBJETIVOS

A pesquisa teve como objetivo geral avaliar os resultados de desempenho de uma rede SDN em situação de sobrecarga na interface do

controlador e impacto de um mecanismo de descarte de pacotes. Para tanto, foram definidos os seguintes objetivos específicos:

A. Mensurar o comportamento de rede SDN com sobrecarga e sem o mecanismo a partir de testes de transmissão *User Datagram Protocol* (UDP) e *Transmission Control Protocol* (TCP).

B. Replicar o mecanismo BH em ambiente emulado.

C. Mensurar o comportamento da rede SDN com sobrecarga e o mecanismo a partir de testes de transmissão UDP e TCP.

D. Realizar análise comparativa dos dois cenários avaliados.

A realização dos objetivos dessa pesquisa deve contribuir para a avaliação dos cenários de sobrecarga que carecem de solução e do planejamento de cenários que poderiam ser beneficiados pela utilização do BH.

A seguir, no capítulo 2, há uma revisão dos conhecimentos para esta pesquisa e também avaliação de pesquisas relacionadas. No capítulo 3 é descrita a metodologia utilizada e, no capítulo 4, são apresentados os resultados com suas interpretações. Por fim, no capítulo 5 está a conclusão do que foi realizado.

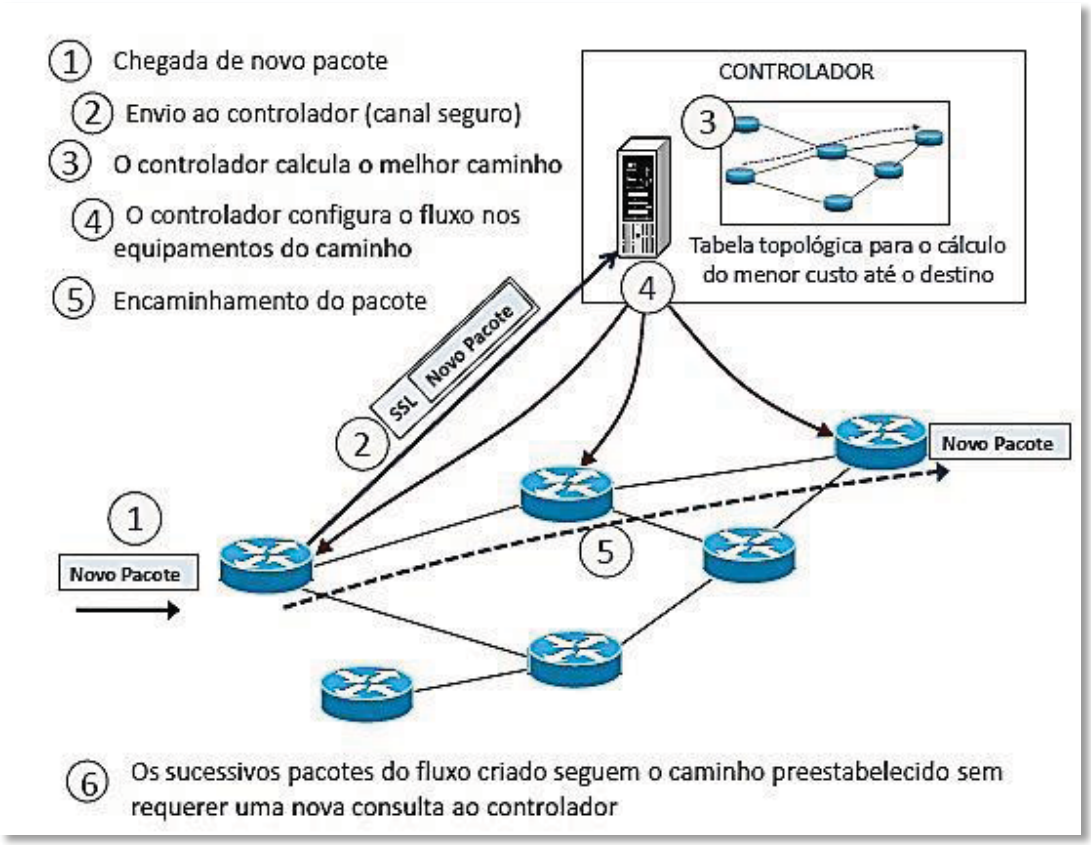
2 REVISÃO DOS CONHECIMENTOS

Este trabalho tem como ponto de partida a utilização de redes definidas por software (*Software Define Network* - SDN) para serviços de transmissão sob demanda devido a suas exigências de qualidade, seus conhecimentos em estabelecimento de rotas nessas e sua possibilidade de emulação de sua estrutura. Tais serviços, geralmente, utilizam o protocolo de transporte UDP que permite um processamento mais rápido que outros por não prever confirmação de recebimento e, consequentemente, sem retransmissão. Em redes tradicionais as rotas são estabelecidas manualmente por um responsável ou por um protocolo de roteamento como *Open Shortest Path First* (OSPF) (TANENBAUM, 1997). Protocolos de roteamento consultam periodicamente as redes que os equipamentos diretamente conectados conhecem e as registram numa tabela – em SDN esse papel é do controlador.

A Figura 2 demonstra a estrutura e funcionamento básico de uma SDN que utiliza um dos principais protocolos de SDN, o OpenFlow (ONP, 2012). Toda configuração é feita a partir do controlador. Se o switch recebe um pacote endereçado para um destino desconhecido, ele solicita ao controlador a definição de qual deve ser a ação (normalmente recebe a porta de destino para repasse do pacote). Essa centralização facilita a configuração, porém, se há falha nesse controlador, toda a rede pode ficar comprometida. O equipamento só é independente para encaminhamento de pacotes para destinos conhecidos e utilizados recentemente (RODRÍGUEZ, 2014).

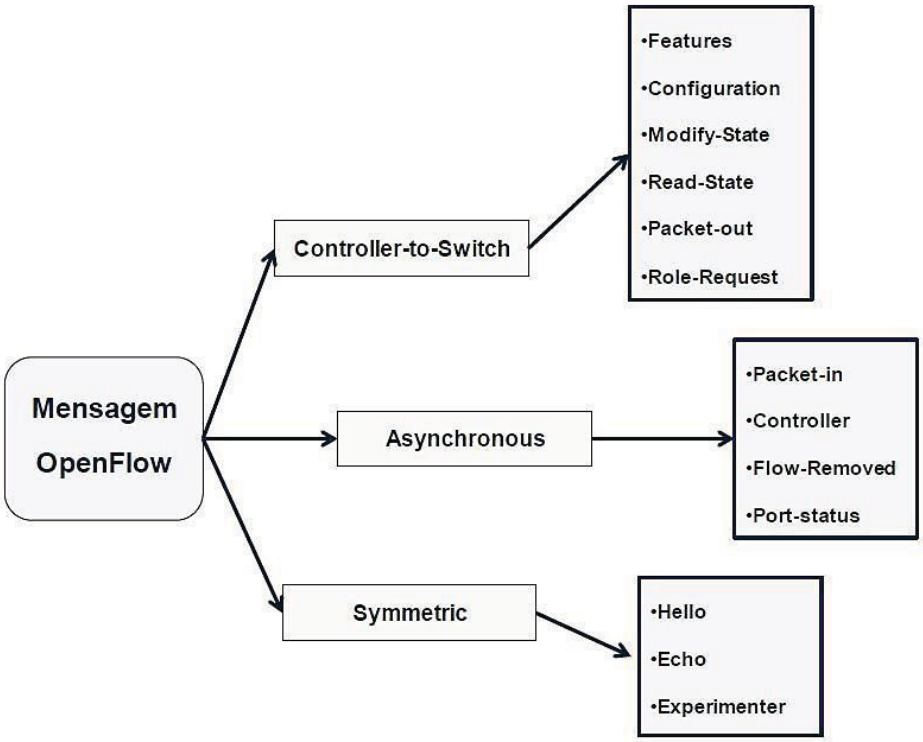
Em SDN alguns protocolos podem ser utilizados na comunicação entre Controlador e elementos de rede, para envio de comandos do controlador e envio de informações dos elementos. O protocolo utilizado nesta investigação foi o OpenFlow, por ser OpenSource e o mais utilizado. Porém existem outros, como Open Virtual Switch DataBase (OVSDB), e proprietários de marcas, como Cisco (OpFlex). A versão mais atual do OpenFlow é a 1.6 de setembro de 2016, no entanto a mais utilizada ainda é a 1.3, de 2012. Os pacotes OpenFlow utilizam o TCP na camada de transporte. São treze tipos de mensagens, conforme a Figura 3.

Figura 2 – Conexões físicas e processo de encaminhamento em SDN com Openflow



Fonte: RODRÍGUEZ, 2014.

Figura 3 – Mensagens do OpenFlow



Fonte: RODRÍGUEZ, 2014.

O switch deve ser um equipamento habilitado para utilizar o mesmo protocolo de comunicação com o controlador. Ele pode ser um simples switch capaz de seguir apenas os comandos do controlador para atualizar a tabela de fluxos ou um roteador com outras funcionalidades. Há vários fabricantes de equipamentos habilitados para SDN, como HP (exemplo, série FlexFabric 5930, com até 96 portas de 10 Gbps), Intel (exemplo, série Ethernet Switch FM6000, com até 64 portas de 10 Gbps), Juniper (exemplo, roteadores da série PTX, com módulos até 24 portas de 40 Gbps) e Cisco (exemplo, série Catalyst 3850, com módulos até 48 portas de 1 Gbps). Neste estudo foi utilizado o Open vSwitch (OVS), que é um switch virtual e flexível quanto aos recursos.

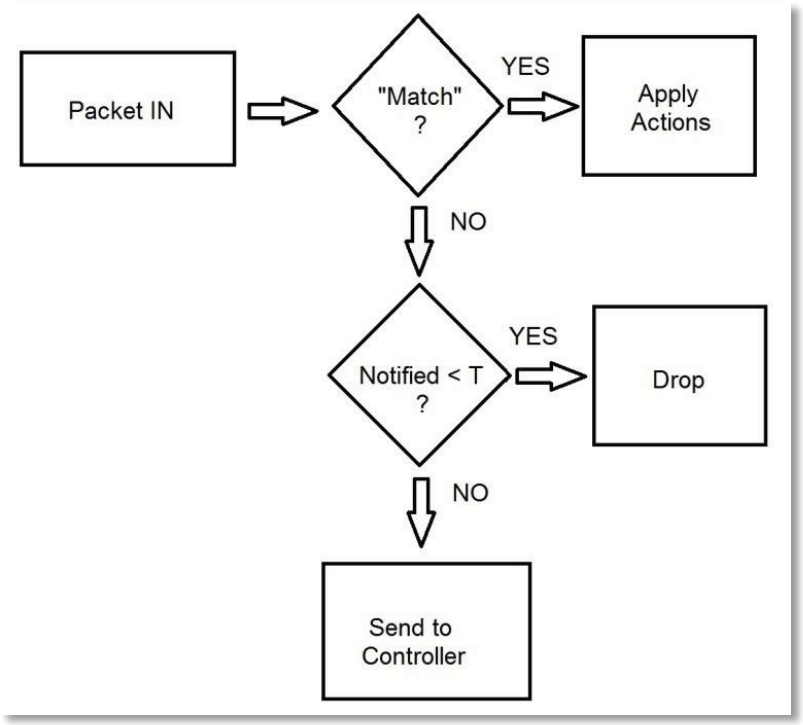
O controlador é o responsável pela configuração dos switches. Existem os controladores proprietários, como os da Cisco, da Juniper e da HP, e existem os abertos, como é o caso do POX (desenvolvido em Python), do RYU (Python), do ONOS (Java) e do OpenDaylight (Java). No controlador podem ser implementadas várias funcionalidades por meio das aplicações, como sistemas de segurança, controle de fluxo de dados e gerenciamento da rede.

Ainda no controlador é possível configurar parâmetros como: Soft Timeout, que remove rotas da tabela de fluxos após um determinado período de tempo sem utilização; Hard Timeout, que força a remoção de rotas da tabela de fluxos do switch em um prazo de tempo determinado, independentemente da utilização; Ação primária, que mostra quais ações o switch deve tomar ao receber pacotes para um destino desconhecido. A ação primária, se não configurada, é definida como o envio dos pacotes para o controlador até que o controlador informe qual deve ser a ação para aquele fluxo. Após a resposta do controlador, o switch atualizará sua tabela e realizará ações, como o encaminhamento para uma porta ou o descarte. A ação primária pode ser configurada como qualquer outra ação, inclusive realizando mais de uma ação.

Esses recursos permitem implementar a proposta de descarte de pacote. A Figura 4 apresenta o processo de funcionamento do mecanismo. A solução reduz o risco de congestionamento da interface, porém, causa o descarte de parte dos pacotes enviados.

Além de atuar no funcionamento do sistema, existem várias formas de realizar a coleta de dados estatísticos do tráfego. Um trabalho, de 2016, de Muragaa, Seman e Mahusin, apresentou um modulo para coleta de dados em que um controlador solicita informações dos switches a cada 5 segundos. Ele utilizou o Mininet para realizar testes e comparou suas medições com as medições da ferramenta Wireshark.

Figura 4 – Fluxograma do mecanismo Blackhole



Fonte: FAVARO, 2015.

2.1 TRABALHOS RELACIONADOS

O objetivo de mitigar o problema do congestionamento na interface entre Switch-Controlador SDN (S-C), seja diminuindo a dependência do controlador pelo switch ou alterando a forma como eles se comunicam, foi tema de outros trabalhos anteriores. Em geral, diminuir a dependência do controlador se distancia da essência do SDN, pois implica remover a facilidade de configurar vários equipamentos em um único ponto. Alterar a forma como eles se comunicam parece ser mais adequado. Ainda não há um padrão quanto aos casos de congestionamento nessa interface, e mais estudos, como os seguintes, precisam ser feitos para melhores definições.

Mogul *et al*. (2010) implementaram uma versão modificada do Openflow, chamada DevoFlow, a fim de diminuir custos de comunicação na rede e processamento do controlador e do switch. Uma das modificações foi delegar para o switch parte das decisões de encaminhamento de novos fluxos e, assim, reduzir tanto a dependência do controlador como o tráfego na S-C. Dessa forma, o controlador tem mais recursos para funções mais importantes, mas de outro lado não tem domínio total sobre a tabela de fluxos. Essa solução seria

interessante para situações em que a demanda do controlador é maior que sua capacidade em definitivo, porém, como solução temporária, pode ser de execução complexa.

Kim *et al*. (2013), apesar de focarem-se no gerenciamento de falhas em redes SDN, propuseram a utilização de um algoritmo de fluxo rápido de configuração (*Fast Flow Setup* - FFS) que reduz a troca de pacotes entre equipamento e controlador nos cenários de múltiplas falhas. O modelo prevê a detecção, o diagnóstico e a correção ou o contorno de falhas de modo contínuo. Sendo um sistema automático, descentralizado e pouco determinístico. Ele emprega processos cognitivos como reação (reação instantânea a eventos), deliberação (escolha do melhor plano disponível) e reflexão (gestão do processo de deliberação). Uma representação simples de aplicação de cognição é a criação de vários agentes capazes de trocar mensagens para tomada de decisão, ou reação, sobre um evento.

Restauração de falhas em SDN são mais lentas que em redes tradicionais, por dependerem de um controlador que envia um comando por vez. Por esse motivo a maioria dos equipamentos possuem *backups* internos para acelerar restaurações, porém, há falhas que afetam os *backups* e outras que exigem outras ações. O FFS respeita as premissas do SDN e é muito útil em redes de larga escala, em que múltiplas falhas são mais prováveis, contudo, a configuração do gatilho para ele ser executado é complexo e exige um controlador com grande capacidade de processamento.

Também buscando aumentar a tolerância do sistema a falhas e reduzir a necessidade de comunicação entre switch e controlador, Ramos (2013) apresentou uma proposta de implementação de um mecanismo de roteamento baseado em OpenFlow. A proposta consiste em codificar no cabeçalho dos pacotes o caminho que os dados devem percorrer da origem até o destino. Foi realizada a instalação em um ambiente virtual controlado, que demonstrou eficiência na recuperação de falhas e na utilização de múltiplos caminhos ao usar o estado de tabela mínima de fluxos nos equipamentos. Essa solução também respeitou os conceitos do SDN e é mais simples de executar, pois a lógica é sempre utilizada, e ela não precisa de um gatilho, porém se distancia ainda mais de redes tradicionais.

Uma técnica de roteamento por segmento com o objetivo de reduzir o volume de pacotes enviados para o controlador foi avaliada (Thaenchaikun, 2017). A técnica consiste em utilizar códigos quando o fluxo precisar passar por mais de um elemento. O primeiro elemento recebe o código, ou códigos, do controlador. Em cada um é informado o próximo salto. Assim os elementos seguintes não precisam acionar o controlador para encaminhar. Em testes com 2 saltos foi verificado uma redução de 50% no volume de pacotes enviados, em relação ao modelo convencional.

Ataques de negação de serviço podem ocorrer em SDN enviando para um elemento da rede pacotes como diferentes origens para um destino real. Isso faz ele enviar os pacotes para o controlador, que será sobrecarregado. Polat e Polat (2017) avaliaram o impacto desse ataque utilizando Mininet com controladores POX e ODL. Em ataque menor houve redução do volume de tráfego transferidos entre 2 nós e em ataque maior houve interrupção da transferência.

SDN aplicada em redes sem fio também está sujeita a sobrecarga na interface com o controlador. Considerando fatores como perda de cerca de 90% dos pacotes, foi avaliado uma solução de multiplexação do trafego enviado para o controlador (Baddeley, 2017). Ela não prevê a eliminação da sobrecarga, mas garantir que solicitações de todos os elementos cheguem ao controlador. A multiplexação consiste em garantir uma janela de tempo periódica para cada elemento realizar transferência para o controlador. Foi verificado redução do tempo de resposta em comparação ao modelo sem a solução.

Outra abordagem foi apresentada por Mendoza, Férrus e Sallent, em 2017. Eles propuseram um sistema de engenharia de tráfego para redes SDN heterogêneas, terrestres e por satélite. Para garantir a qualidade de serviço, o sistema faz uma série de validações, por exemplo, caso precise de taxa de transferência constante, regras de roteamento diferentes para cada conexão são aplicadas. Mesmo os elementos da rede limitam a admissão de conexões e estabelecem banda máxima para elas. Então, se o sistema considerar que a rede está totalmente ocupada, ninguém mais irá conseguir comunicação, e quem já está usando a rede tem a garantia do serviço não degradar. Essa solução respeita os conceitos do SDN e pode ser implementada utilizando lógicas

simples de ocupação de canal, mas, por outro lado, pode gerar ociosidade da rede.

O mecanismo BH foi selecionado para a pesquisa pois está de acordo com os conceitos de SDN e pouco altera o ambiente da rede. Ele não exige alterações no protocolo SDN e nem de arquitetura da rede, apenas aplica uma regra padrão para novos fluxos no switch, podendo ser implementado diretamente no switch ou por comando do controlador.

O próximo capítulo apresenta como esses fundamentos foram utilizados na pesquisa e, na sequência, os resultados obtidos e sua avaliação.

3 METODOLOGIA

Os testes da pesquisa foram planejados para três etapas: (A) avaliar se o ambiente emulado correspondia a realidade; (B) avaliar as consequências do congestionamento na S-C; e (C) avaliar as consequências da utilização do BH.

A infraestrutura utilizada foi composta por:

- Servidor Debian versão 9 (stretch);
    - **a.** Processador de 4 núcleos a 3,2GHz;
    - **b.** Memória RAM de 10GB.
- Máquina virtual Ubuntu 14.04 em notebook;
    - **a.** Processador de 2 núcleos a 2,7GHz;
    - **b.** Memória RAM de 3GB.
- Emulador Mininet versão 2.0.1+;
- Switch Open vSwitch 2.5.1;
- Controlador POX;
- Protocolo de rede OpenFlow 1.0;
- Configurador de interfaces de rede TCConfig;
- Gerador de tráfego IPERF 2.0.5, pacotes de 1500 bytes;
- Scripts em linguagem Python 2.7.12;
- Analisador de protocolos Wireshark 1.12.1.

O POX foi usado neste trabalho por já ser amplamente utilizado em pesquisas e ter uma estrutura lógica de fácil interpretação. Ele teve sua última atualização em 2013. Pode rodar sobre Linux, Windows ou Mac OS. Ele possui suporte para Open vSwitch e extensões da Nicira (desenvolvedor original) (MCCAULEY, 2015).

Uma das extensões da Nicira, “Learn”, permite que o switch utilize as informações dos pacotes para realizar ações e não apenas encaminhar essas informações para o controlador. Assim é possível comparar endereço de origem, endereço de destino, protocolo de transporte e outros dados do pacote com padrões esperados para o switch reagir rapidamente, sem necessariamente acionar o controlador (OPENVSWITCH, 2018).

O emulador Mininet possui recursos para facilitar a criação de arquiteturas de SDN com Openflow e, assim, criar redes virtuais realísticas. Ele também facilita a automatização de testes com *scripts* em linguagem Python (MININET, 2018). Ele precisa ativar equipamentos virtuais para funcionar, como um switch Open vSwitch (OVS), e permite a comunicação desses com equipamentos reais. Podendo, assim, escolher entre utilizar um controlador embarcado ou um controlador real (remoto). Contudo, ele não oferece controle sobre a interface do switch com o controlador e nem é capaz de simular a geração de tráfego. Fazendo necessário a utilização de recursos do sistema operacional, como o TCconfig no Linux (HOMBASHI, 2018) e o Iperf2 (MCMAHON, 2018). Apesar de permitir utilização mista de equipamentos virtuais e reais, o Mininet possui limitações de desempenho abaixo de outros simuladores (WANG, 2014).

A topologia inicial foi de um switch com um controlador e dois terminais, conforme Figura 5, que é suficiente para validar o funcionamento geral e causa o menor impacto no consumo de recursos compartilhados para emulação. Esses testes foram executados no notebook, e os posteriores, topologias com mais terminais de origem, no servidor conforme Figura 6. O código de controle utilizado foi baseado no “forwarindg.l2_learning.py” fornecido pelos desenvolvedores do controlador, que pode ser verificado no Apêndice 1. Nele foi alterado apenas a configuração do Hard Timeout (HT), que é o tempo máximo que um registro de encaminhamento fica na tabela de fluxo, mesmo que ele esteja em uso. Ele ficou definido em 30 segundos, para permitir várias interrupções numa mesma transferência e num intervalo que uma não interfira na outra.

Figura 5 – Topologia Inicial da rede

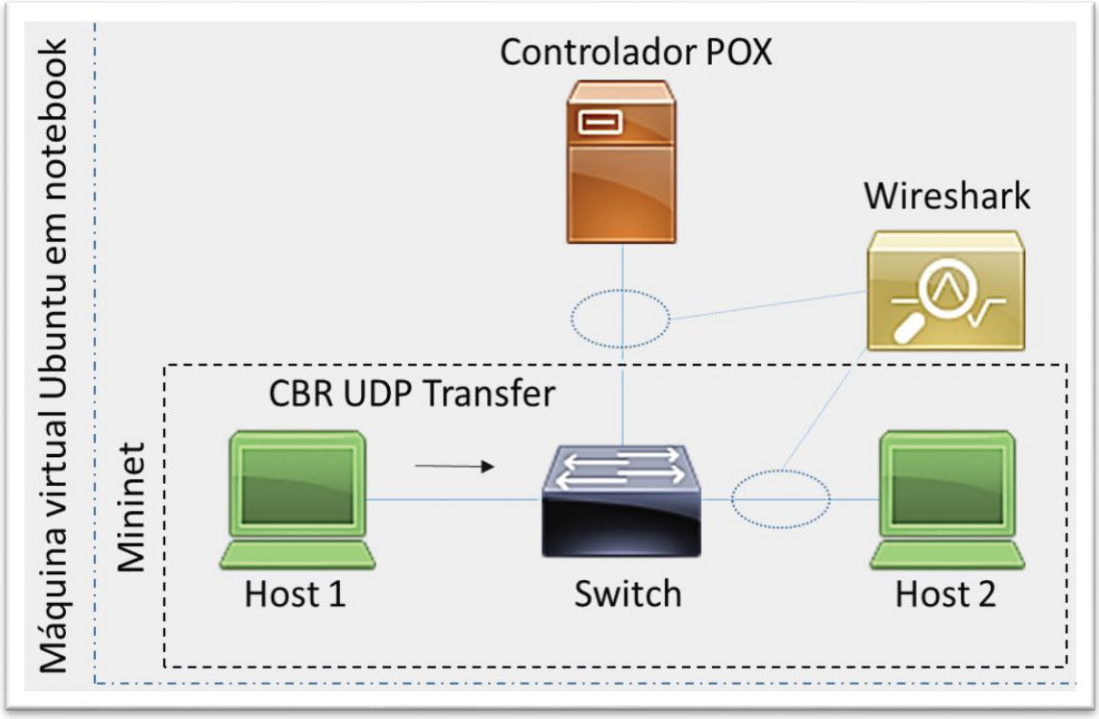
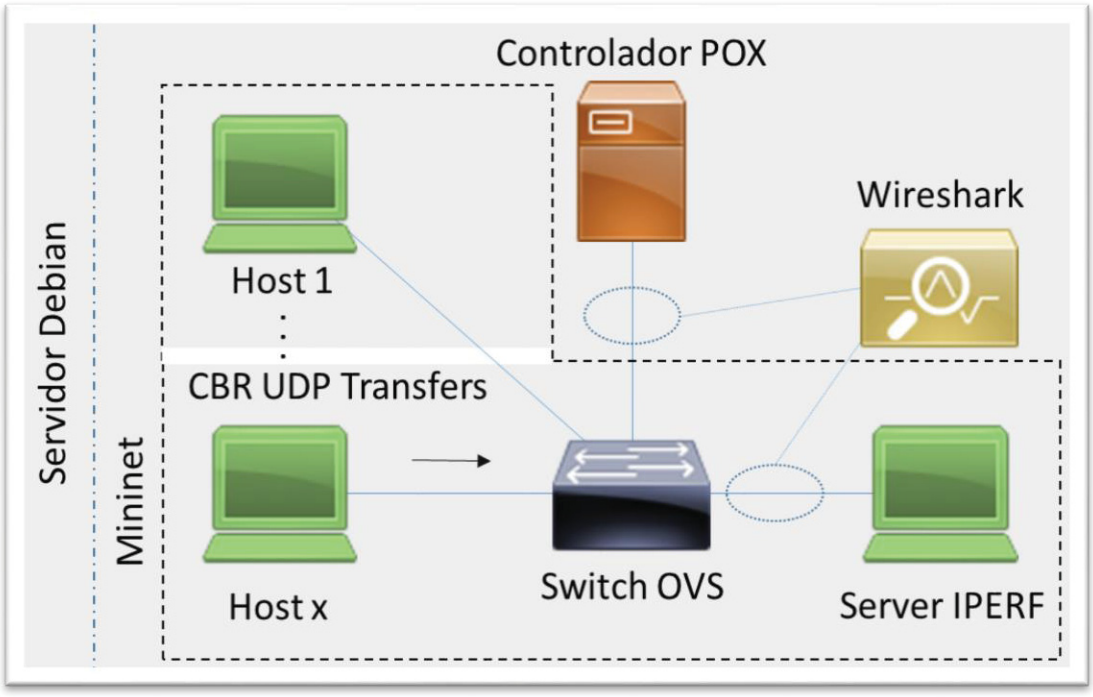


Figura 6 – Topologia dos testes com maior estrutura



Foi utilizado o aplicativo IPERF para gerar tráfego utilizando os protocolos de transporte User Datagram Protocol (UDP) e Transmission Control Protocol (TCP) entre os terminais e medir a quantidade de pacotes recebidos no

terminal de destino. Foram registradas e armazenadas medições consolidadas a cada um segundo de transferência, para comparação com o volume enviado. Também foi utilizado o Wireshark como ferramenta de coleta de dados (pacotes) e exportação para o formato de dados separados por vírgula (*Comma-Separated Values* - CSV). A ferramenta coletou pacotes dos dois sentidos da interface entre controlador e switch, que para o sistema operacional é a porta *loopback*, e da interface entre switch e terminal de destino.

Foi definida uma sequência de passos para iniciar o ambiente e realizar os testes, conforme listado abaixo. Ela foi seguida mesmo com a automação dos testes, cujo código pode ser visto no Apêndice 2.

1. Ativação do controlador. Nesse ambiente, foi iniciado o POX em um terminal de comando do sistema operacional, e como parâmetro foi definida a lógica de encaminhamento “forwarding.l2 learning”.

2. Criação da topologia. Nesse ambiente, foi iniciado o Mininet em outro terminal, e como parâmetro foi definida a topologia desejada. Um switch OVS e pelo menos dois hóspedes.

3. Configuração do switch para aplicar o Blackhole nos testes correspondentes.

4. Definição da banda limite entre controlador e switch. Nos testes de comparação por velocidade não houve limitação, apenas nos testes de estrangulamento da interface e utilizando o TCconfig.

5. Início da coleta de pacotes da interface entre switch e futuro servidor IPERF pelo Wireshark.

6. Início da coleta de pacotes da interface do controlador pelo Wireshark.

7. Ativação de servidor IPERF. Nesse ambiente foi ativo no hospede conectado à porta monitorada do switch.

8. Ativação de clientes IPERF. Nesse ambiente foi ativo um cliente IPERF em cada hóspede restante para iniciar a transferência de

pacotes para o servidor. Nesse comando são permitidas várias customizações, como IP de destino, protocolo de transporte, porta de destino, frequência de apresentação de resultados, tempo de transferência em segundos e velocidade.

9. Acompanhamento da execução. A finalização da transferência é informada no terminal do cliente IPERF.

10. Exportação dos dados. Finalizada a coleta de pacotes e exportação dos dados no formato CSV para análise.

Os testes de UDP sem implementação do Blackhole (BH) para medição da quantidade de pacotes de controle foram realizados utilizando oito velocidades diferentes: 100 Kbps, 1 Mbps, 5 Mbps, 10 Mbps, 40 Mbps, 50 Mbps, 60 Mbps e 100 Mbps. Para TCP: 100 Kbps, 200 Kbps, 500 Kbps, 1 Mbps, 10 Mbps, 20 Mbps, 50 Mbps e 100 Mbps. A configuração para cada velocidade acontece ao editar o comando do passo 8. Para cada velocidade foram realizadas três execuções com duração de 300 segundos, totalizando trinta eventos por velocidade, para confirmar o padrão de comportamento. Também foi avaliado o limite de banda para congestionamento da interface com uma transferência na velocidade de 10 Mbps, para isso foi adaptado os comandos do passo 4 e também foram avaliadas 30 execuções.

Com a implementação do BH foram realizados testes utilizando velocidades de 5 Mbps, 40 Mbps e 100 Mbps. Sendo gerada a mesma quantidade de eventos e da mesma forma. Testes com outras velocidades ou de congestionamento não foram necessários, pois os resultados se apresentaram estáveis na comunicação controlador-switch. Então, foram executados testes com e sem o mecanismo, utilizando vários clientes com transferência simultânea de 10 Mbps cada para o servidor, sendo que a interface entre controlador e switch estava limitada a 10 Mbps, 20 Mbps ou 40 Mbps. Para o limite de 10 Mbps foram de 2 a 12 clientes, para o de 20 Mbps foram de 26 a 50 e para o de 40 Mbps foram de 64 a 98, sendo consideradas apenas quantidades pares.

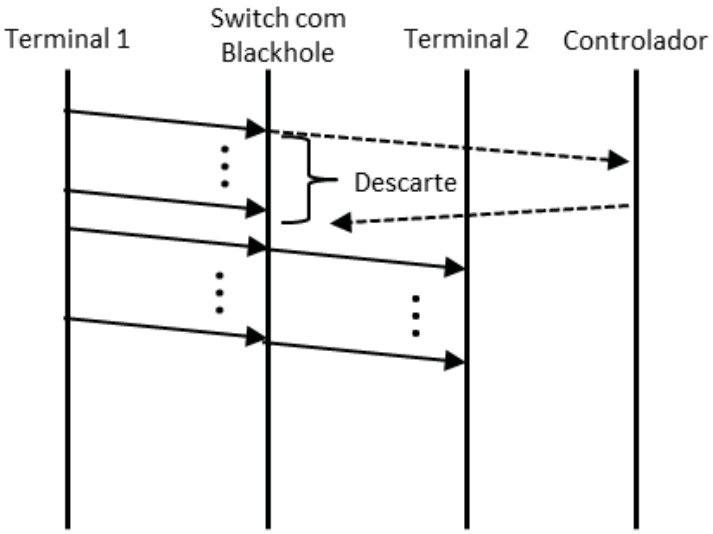
O simples aumento da banda de transferência numa rede SDN já é suficiente para aumentar o volume de pacotes enviados ao controlador. A escolha em aumentar o número de clientes em vez de aumentar a banda se deve

ao fato de que mais clientes significam mais comandos do controlador para atualizar a tabela de fluxos do switch. Para esta pesquisa, essa situação é mais rica porque gera evidências de impactos no controle dos fluxos.

Cada execução gerou dois arquivos no formato CSV, originados pelo Wireshark, contendo as seguintes informações: número sequencial dos pacotes, tempo em que o pacote foi gerado em relação ao início da coleta, endereço IP de origem, endereço IP de destino, protocolo da camada superior, tamanho do pacote, campo de identificação do pacote do protocolo IP e descrição do pacote (info). Além de um arquivo CSV gerado pelo IPERF contendo quantidade de pacotes enviados, perdidos e fora de ordem a cada segundo.

Para a configuração do BH foi utilizando a extensão “Learn” do OVS na regra padrão, em que para qualquer fluxo novo eram aplicadas duas ações, envio do pacote para o controlador e aplicação de regra de baixa prioridade para descarte de pacotes com aquela característica por um segundo, conforme Figura 7. Sendo assim os pacotes seguintes daquele fluxo deveriam ser descartados pela regra até que o switch aplicasse a nova regra enviada pelo controlador. Essa regra para descartar tem baixa prioridade, arbitrariamente definida em 100, para não sobrepor a regra que é enviada pelo controlador para o correto tratamento do pacote, que tem prioridade, geralmente, de valor maior que 6000. O tempo foi definido em 1,5 segundos para se ter uma para margem de segurança, mas seguramente poderia ser avaliado a aplicação de um valor menor.

Figura 7 – Comportamento esperado a cada HT



A automatização dos testes foi envolvendo as partes do Mininet, incluindo switch e terminais. As tarefas envolvendo ativação do controlador (passo 1) e o Wireshark (passos 5, 6 e 10) foram realizadas manualmente. Todas as outras tarefas podiam ser executadas pelo *script* em Python conforme ele era parametrizado.

As informações coletadas serviram para consolidar as medições listadas abaixo, as quais permitiram realizar as comparações de desempenho. As comparações realizadas foram para as diversas taxas de transferência e os diversos limites na interface com o controlador, com ou sem o BH.

Quantidade de pacotes por segundo na interface entre switch e controlador:

- Quantidade de pacotes por segundo retransmitidos para o controlador na ocorrência de novo fluxo (ou HT);
- Quantidade de pacotes por segundo descartados na transmissão para o controlador na ocorrência de novo fluxo (ou HT);
- Quantidade de pacotes fora de ordem na ocorrência de novo fluxo (ou HT);
- Tempo médio para restauração de rota na ocorrência de novo fluxo (ou HT);

- Quantidade de bytes transmitidos para o controlador na ocorrência de novo fluxo (ou HT);
- Quantidade de bytes recebidos pelo servidor IPERF na ocorrência de HT.

4 RESULTADOS

Primeiramente foi verificado o comportamento do ambiente da rede e sem realização de descarte. Como resultado do teste sem limite de banda, houve coleta de pacotes de duas situações na interface do controlador, o que era esperado por ser um ambiente controlado, pacotes do teste de comunicação e de identificação de novo fluxo após um Hard Timeout (HT) seguido da definição de restauração. O teste de comunicação é uma verificação de que a interface está funcionando, o que ocorre a cada cinco segundos, em que o controlador envia uma pequena mensagem num pacote OpenFlow (OFPT_ECHO_REQUEST) e aguarda uma resposta (OFPT_ECHO_REPLY) que irá confirmar o funcionamento. A identificação de novo fluxo ocorre pelo envio de pacotes (PACKET_IN) do switch para o controlador – cujo encaminhamento o switch não sabe – e ocorre até que o controlador informe ao switch qual ação realizar com esses pacotes.

Conforme foi configurado, o switch removia fluxos a cada 30 segundos da tabela devido ao HT. Então, no período de 300 segundos de cada transmissão, havia dez momentos em que o switch enviava os pacotes recebidos do cliente IPERF para o controlador. Ao receber o primeiro pacote, o controlador identificava a ação e respondia ao switch com um pacote de configuração (FLOW_MOD), além de reencaminhar os pacotes recebidos (PACKET_OUT). Com a nova configuração, o switch realizava a ação definida e não enviava mais pacotes para o controlador até o próximo HT. A seguir são detalhados os testes, analisando uma transmissão sem concorrências e, na sequência, transmissões concorrentes.

4.1 TESTES DE ÚNICA TRANSFERÊNCIA

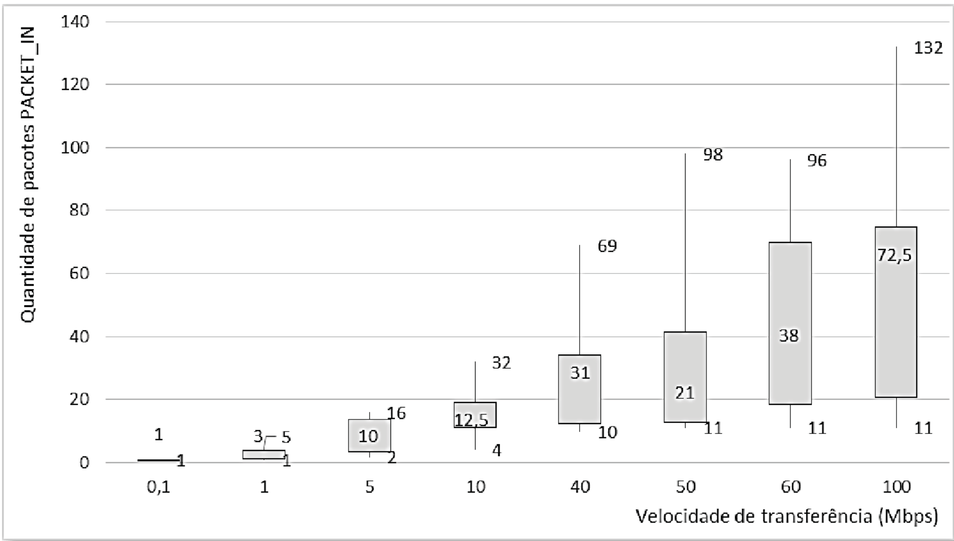
O primeiro dado analisado foi o volume de pacotes gerado na interface entre Switch e Controlador SDN (S-C) para diferentes velocidades. A Tabela 2 informa a quantidade de pacotes PACKET_IN gerados por evento em metade dos testes das velocidades de 1 Mbps, 10 Mbps e 40 Mbps para UDP, na qual se pode observar que há um comportamento para cada taxa de transferência, isto é, a quantidade de pacotes por evento é semelhante para mesma taxa.

Ao comparar os resultados de velocidades diferentes, foi verificado que quanto maior a velocidade, maior é a quantidade de pacotes enviados ao controlador e pouco varia o tempo de restauração do fluxo. A Figura 8 demonstra a mediana de pacotes UDP enviados para o controlador antes da restauração para cada velocidade, além dos máximos, dos mínimos e das concentrações de 50%.

Tabela 2 – Quantidade de pacotes UDP enviados para controlador por restauração

Evento	1Mbps	10Mbps	40Mbps
1º	5	20	14
2º	2	14	35
3º	5	32	47
4º	4	11	31
5º	1	20	10
6º	3	11	31
7º	3	4	11
8º	1	11	32
9º	3	11	69
10º	1	16	12
11º	1	12	14
12º	3	31	26
13º	5	8	31
14º	4	24	35
15º	2	5	31

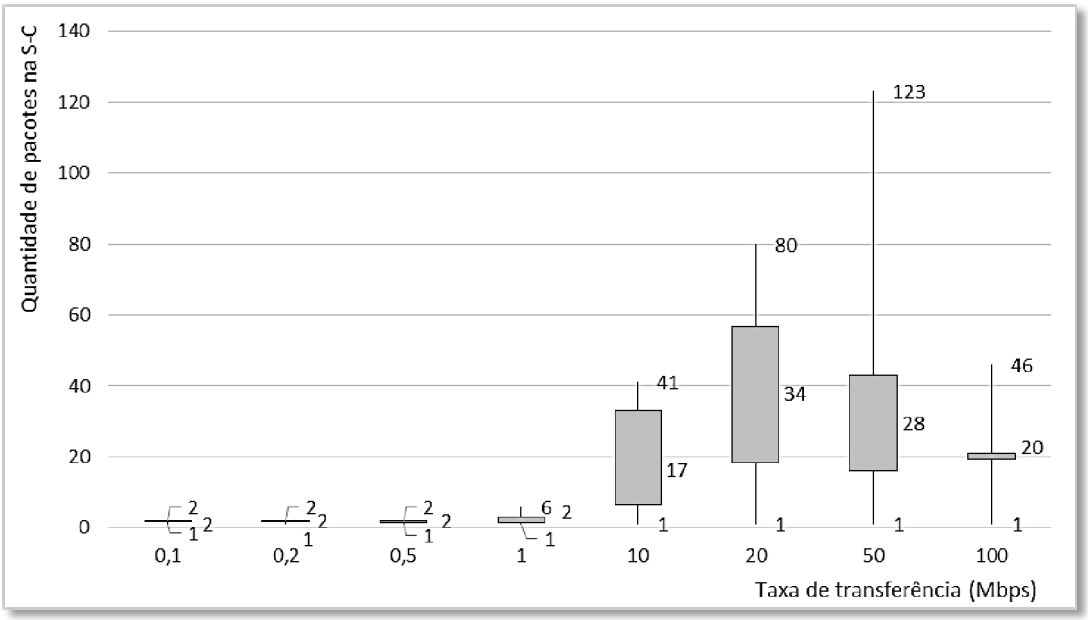
Figura 8 – Volume de pacotes PACKET_IN enviados para o controlador por taxa de transferência para UDP



Para Transmission Control Protocol (TCP) foi observado uma dispersão maior nos resultados, mas ainda com maior chance de aumentar o volume de pacotes enviados para o controlador conforme aumento da banda. A

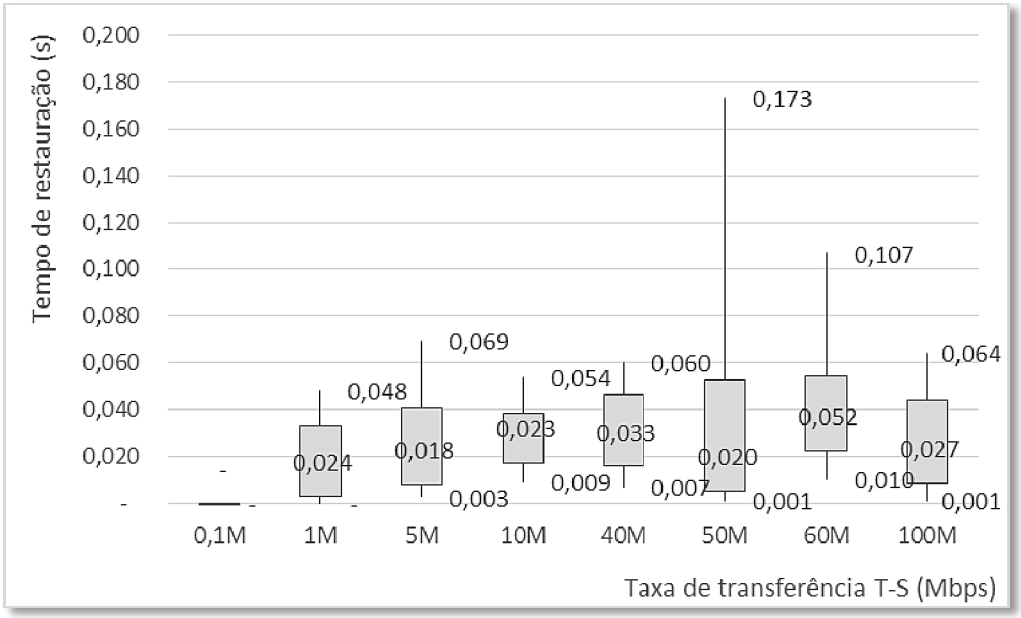
Figura 9 informa o volume de PACKET_IN enviado por taxa de transferência com TCP, assim como a Figura 8 para UDP.

Figura 9 – Volume de pacotes PACKET_IN enviados para o controlador por taxa de transferência para TCP



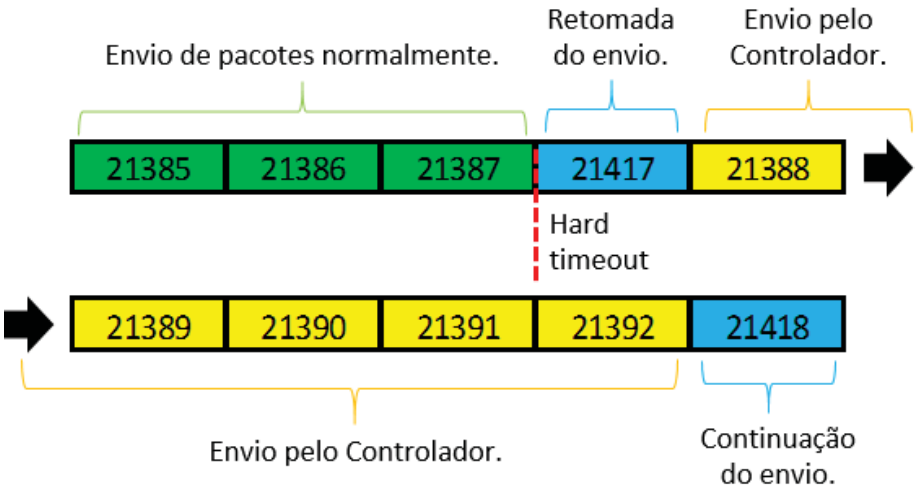
Já a Figura 10 demonstra o tempo mediano para restauração do fluxo no mesmo formato para UDP. O tempo de restauração foi considerado como o tempo entre o envio do primeiro PACKET_IN ao controlador e o envio do FLOW_MOD para o switch. Não houve variação considerável entre bandas diferentes e o TCP apresentou resultado semelhante.

Figura 10 – Tempo de restauração do fluxo por taxa de transferência UDP



Ao analisar os pacotes enviados pelo switch na interface com o servidor IPERF é possível verificar que a cada evento do HT havia pacotes fora de ordem. Isso também era esperado, pois o controlador reencaminha os pacotes recebidos após informar a ação para o switch e, nesse momento, o switch já atuou para encaminhar os novos pacotes recebidos desse fluxo. A Figura 11 demonstra como ocorre o envio dos primeiros pacotes após o HT.

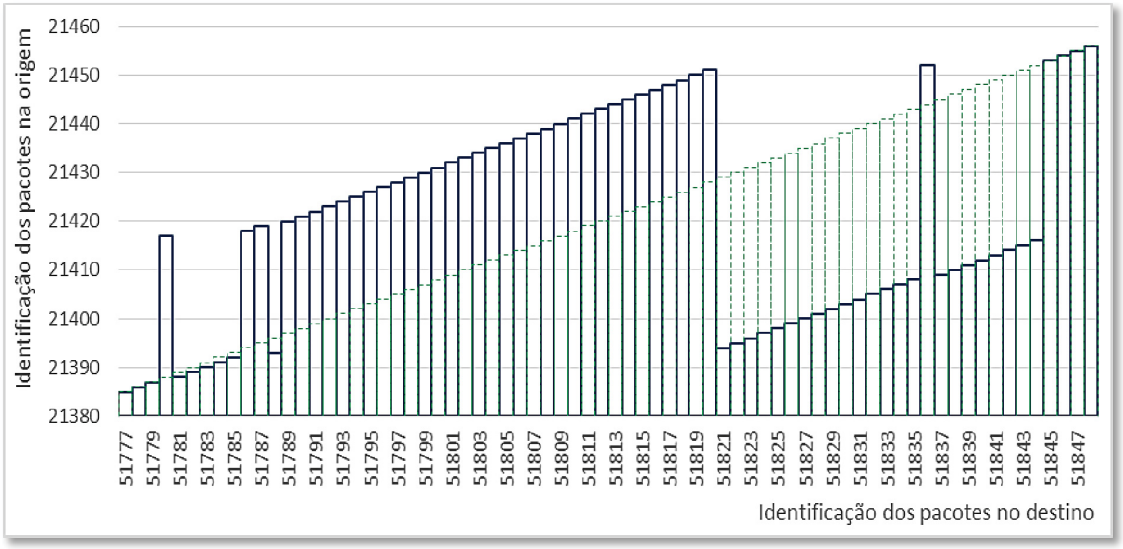
Figura 11 – Pacotes UDP recebidos pelo destino fora de ordem devido a HT



Pacotes recebidos fora de ordem podem afetar o serviço a qual se destina, caso o destino não tiver capacidade de reordená-los. Em um dos

eventos do teste da velocidade de 10 Mbps, houve pacotes recebidos fora de ordem e a ordem apenas foi reestabelecida após 65 pacotes. Os detalhes de como ocorreu essa transferência estão na Figura 12, em que há uma comparação com a ordem de pacotes conforme era esperada.

Figura 12 – Ordem de recebimento dos pacotes UDP e expectativa de recebimento dos mesmos

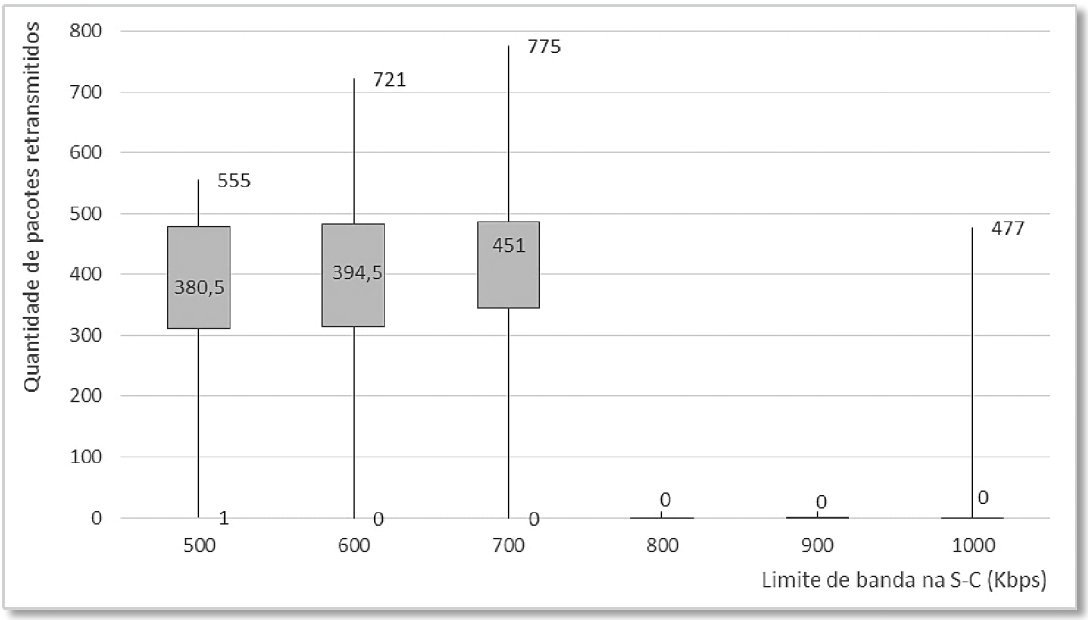


Quando os limites de banda na S-C foram aplicados, houve o surgimento de retransmissão de pacotes, principalmente para os do tipo PACKET_IN. O primeiro teste realizado foi com o limite de 1 Mbps para uma transmissão UDP, pois seria um limite capaz de atender 10% da banda entre terminais e não houve impacto na transmissão. Em sequência, foi testada a metade dessa banda (500 Kbps) e houve uma grande quantidade de pacotes retransmitidos. Assim, ficou claro que a banda necessária para não afetar a transmissão seria entre esses valores, e então foram testados os seguintes limites de banda: 500 Kbps, 600 Kbps, 700 Kbps, 800 Kbps, 900 Kbps e 1 Mbps.

A Figura 13 mostra a comparação entre os limites de banda que foram testadas. Para essa comparação foi medida a quantidade de pacotes retransmitidos na interface S-C para cada limite. Para os limites abaixo de 800 Kbps (8% da banda entre terminais), foi gerado um volume médio muito maior de pacotes retransmitidos, e isso indica a perda de pacotes na transmissão entre terminais. Na retransmissão dos pacotes PACKET_IN, ocorreu o encapsulamento de até vinte pacotes em um único na interface S-C, o que

reduzia a ocupação do canal, mas não o esforço de processamento do controlador.

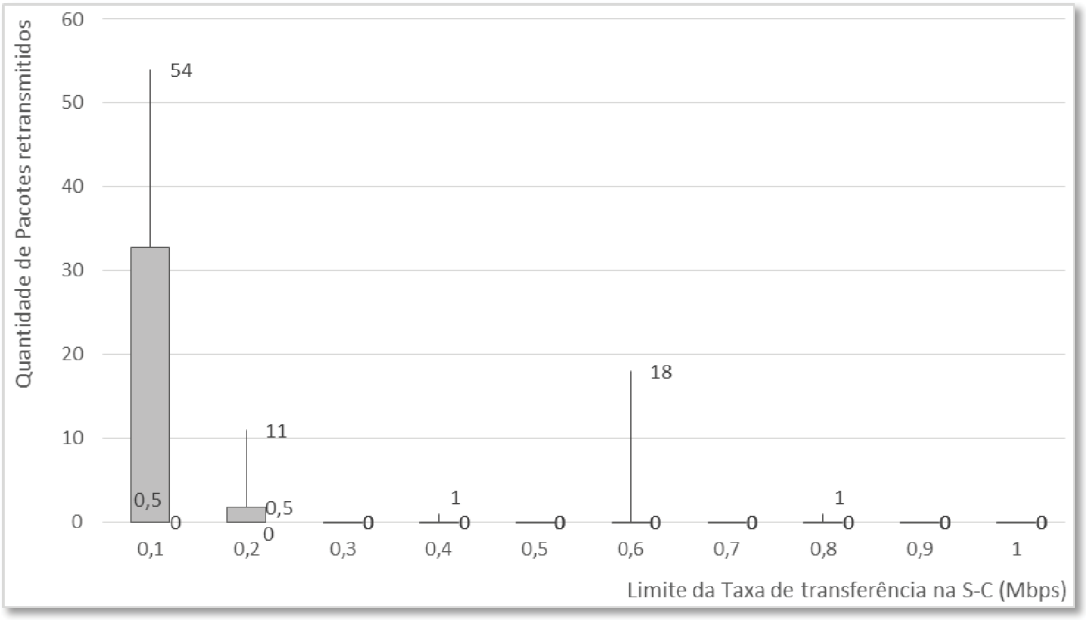
Figura 13 – Quantidade de pacotes UDP retransmitidos a cada HT por limite de banda com controlador



O comportamento da rede nesses cenários nos faz acreditar que se o switch tiver muitos fluxos de alta taxa de transferência, sendo iniciado quase simultaneamente, o serviço do controlador será afetado. O controlador irá receber muitos pacotes de diferentes fluxos ao mesmo tempo, o que irá demandar mais tempo para restauração de cada fluxo. Mais tempo também resultará em receber mais pacotes. A interface pode ficar congestionada por causa disso, o que pode causar o não recebimento de pacotes pelo destino e demandar mais tempo para normalização. Serviços como Video On Demand (VOD) podem perder qualidade.

Já para as transmissões por TCP, houve uma tolerância maior, como é mostrado na Figura 14. Retransmissões significativas ocorreram apenas para limites de 200 Kbps ou menos. Isso ocorre devido ao controle de fluxo realizado pelo TCP, que evita enviar alto volume de pacotes se não há confirmação de recebimento dos enviados anteriormente.

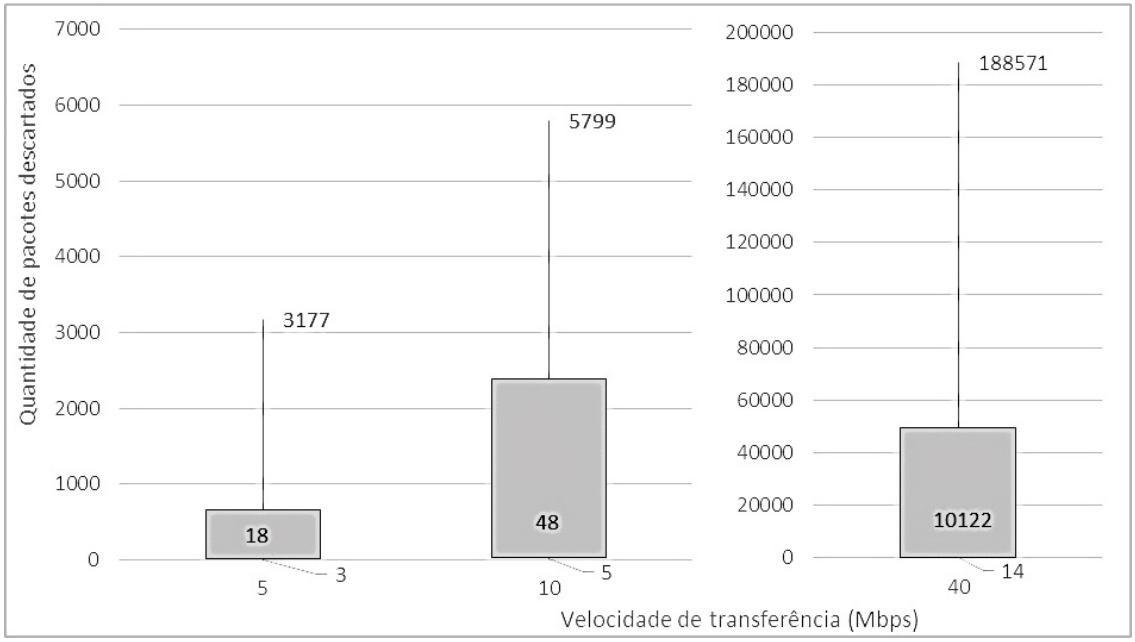
Figura 14 – Quantidade de pacotes TCP retransmitidos a cada HT por limite de banda com controlador



Sabendo que em alguns cenários o volume de tráfego entre controlador e switch pode ser um problema e que ainda não foi desenvolvida uma solução que atenda a todos os cenários perfeitamente, é possível alinhar as necessidades do fluxo às características de cada solução. Um parâmetro que poderia ser utilizado nessa seleção seria o protocolo de transporte, pois possuem necessidades muito diferentes (exemplo: TCP realiza retransmissão, UDP não). A ação padrão de envio de pacotes para o controlador vai garantir o envio deles para o destino se não houver congestionamento.

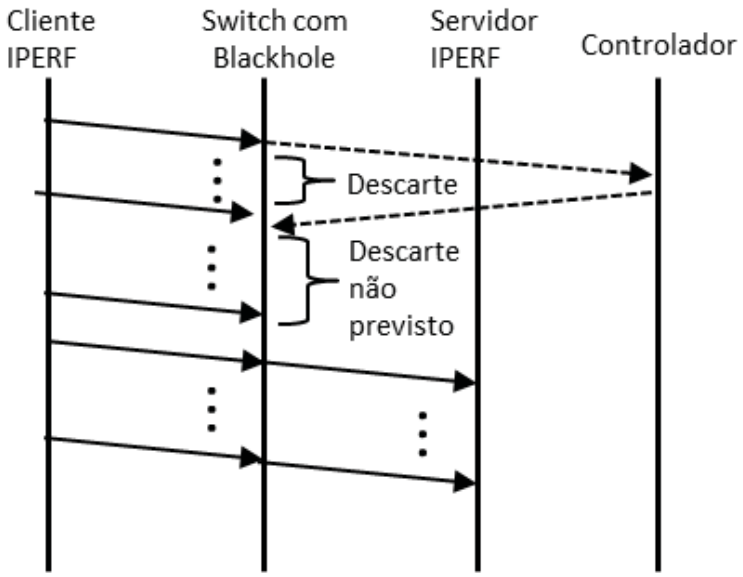
Então, foi avaliado a utilização do BH no mesmo cenário com UDP e confirmou-se que para todos os eventos foi enviado apenas um pacote para o controlador. Os pacotes subsequentes eram descartados até que a ação do controlador fosse aplicada pelo switch. Também não houve pacotes recebidos fora de ordem, apenas variação no volume de pacotes descartados, conforme Figura 15, que monstra a variação para cada velocidade. Esses dados indicam que, em alguns casos, a mediana é próxima à quantidade de pacotes enviados para o controlador sem utilização do BH, como esperado, e, em outros, é maior.

Figura 15 – Quantidade de pacotes UDP descartados a cada HT por taxa de transferência, utilizando BH



Em teste prático, com velocidade de 10 Mbps, foi verificado que há casos em que ocorre o descarte de mais de mil pacotes, sendo que a expectativa era o descarte de doze pacotes (média de envio ao controlador sem BH). Esses casos representam dois a cada dez, já nos outros oito a expectativa é respeitada. A análise dos casos em que a mediana ficou maior apontou que o controlador respondia as solicitações na mesma velocidade que sem o BH pelos registros de transferência. Porém o switch demorava mais tempo para aplicar as ações recebidas do controlador, o que influenciou os resultados dos testes e pode ser considerado um fator externo ao mecanismo. A Figura 16 mostra a situação ocorrida, que é diferente da esperada por descartar pacotes mesmo após resposta do controlador.

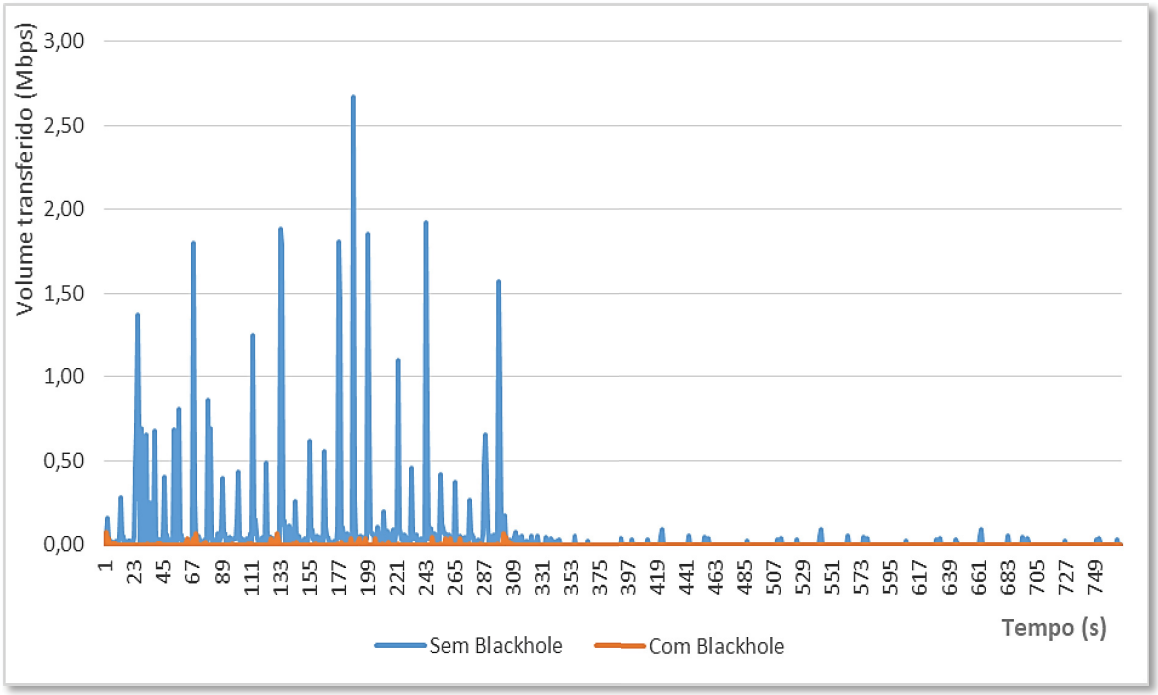
Figura 16 – Estabelecimento de fluxo com o BH



4.2 TESTES DE MÚLTIPLAS TRANSFERÊNCIAS

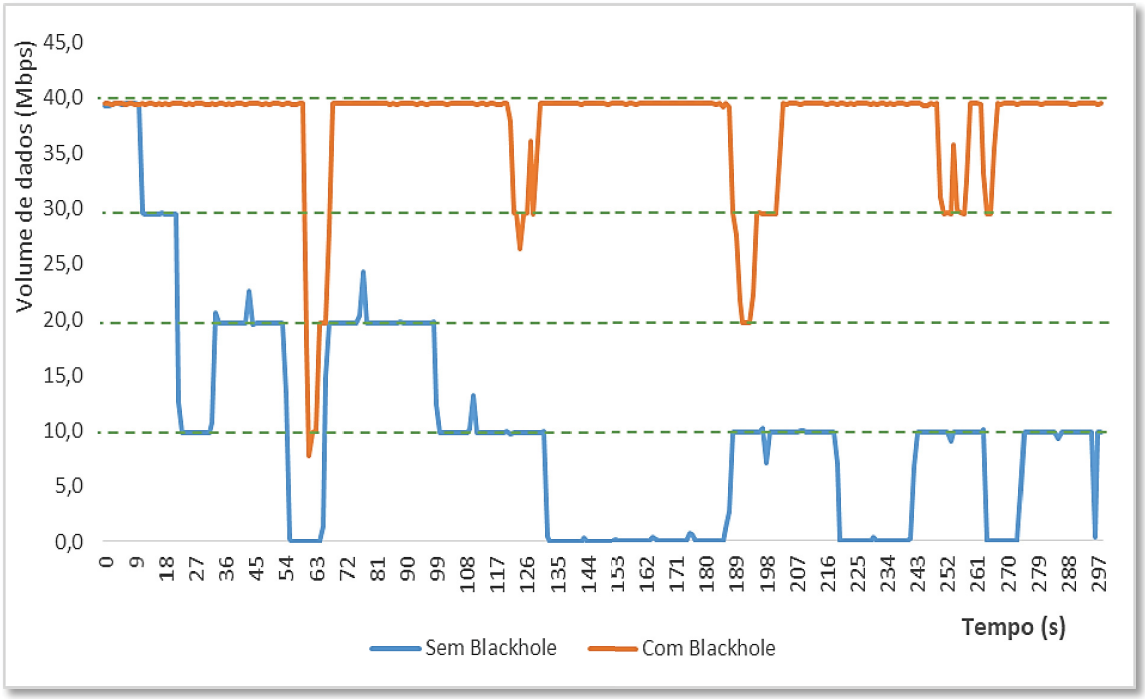
Para avaliar o comportamento do ambiente com uma rede maior foram feitos testes com cinco terminais, sendo quatro clientes e um servidor IPERF. Foi definido um limite de 3 Mbps na interface entre controlador e switch, pois representava menos que 8% da demanda de 40 Mbps utilizando UDP, com a certeza de observar a degradação da comunicação caso o BH não fosse utilizado. Foram realizados os testes nos dois cenários para poder comparar. Foi possível observar uma diferença significativa na ocupação da interface S-C, como pode ser visto na Figura 17. A ocupação da interface S-C chegou próxima ao limite em alguns eventos de HT, quando o BH não era utilizado. Por outro lado, a ocupação permaneceu próxima de zero em todas as situações em que o mecanismo foi utilizado, pois no HT era enviado apenas um pacote para o controlador.

Figura 17 – Volume médio trafegado na interface com controlador limitada a 3 Mbps, com e sem BH



Para o mesmo teste foram analisados os pacotes recebidos pelo servidor IPERF. Cerca de três quartos do volume de dados não foi recebido quando não se utilizou o BH. Já com ele, foram recebidos cerca de 95% dos dados. Na Figura 18 é possível verificar a variação na média de dados recebidos ao longo do tempo de todos os quatro clientes para os dois cenários. A melhor situação é o recebimento de cerca de 40 Mbps de transmissão contínua, porém, isso não ocorre devido aos eventos de HT. O mecanismo BH reduziu significativamente a degradação do recebimento de dados quando comparado ao comportamento padrão da rede.

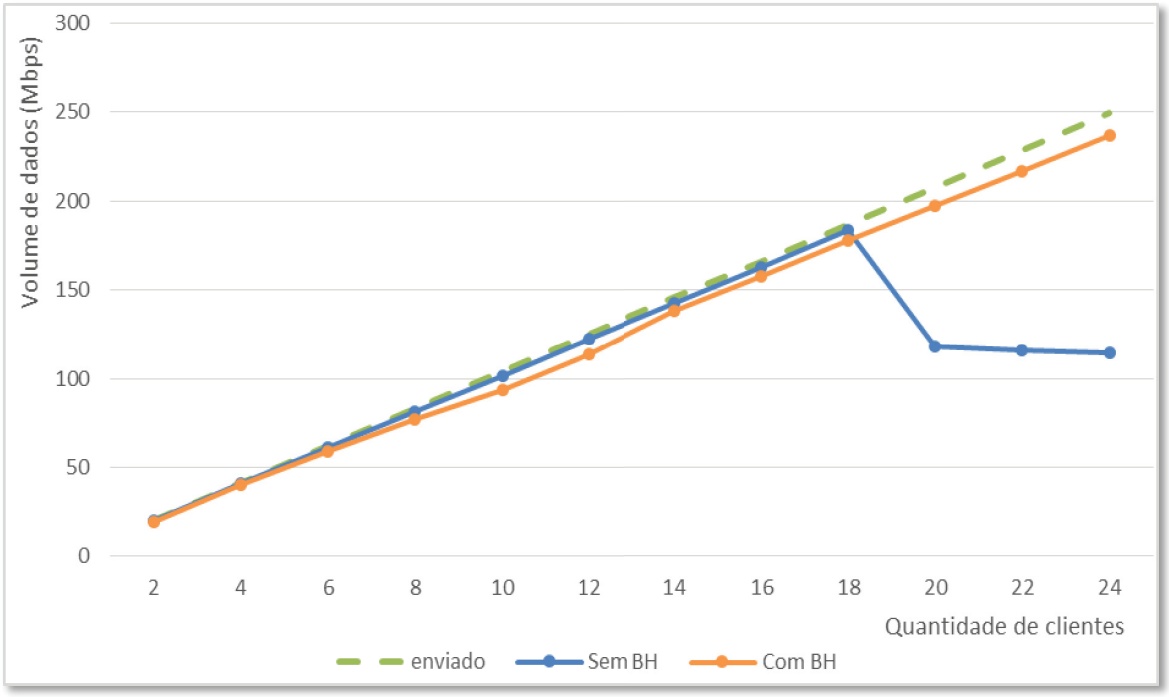
Figura 18 – Média de dados UDP recebidos pelo servidor IPERF por 4 clientes, com o limite de 3 Mbps na S-C



No próximo passo foram realizados testes com topologias maiores, variando o limite de banda na interface S-C e a quantidade de clientes IPERF. Para o limite de 10 Mbps, e utilizando de 2 até 24 clientes, foi possível verificar que o comportamento com e sem BH muda para volumes de tráfego acima de 18 clientes. Com a utilização do mecanismo, o volume de tráfego recebido é um pouco abaixo do enviado. Sem o BH o volume de tráfego recebido é muito abaixo do enviado.

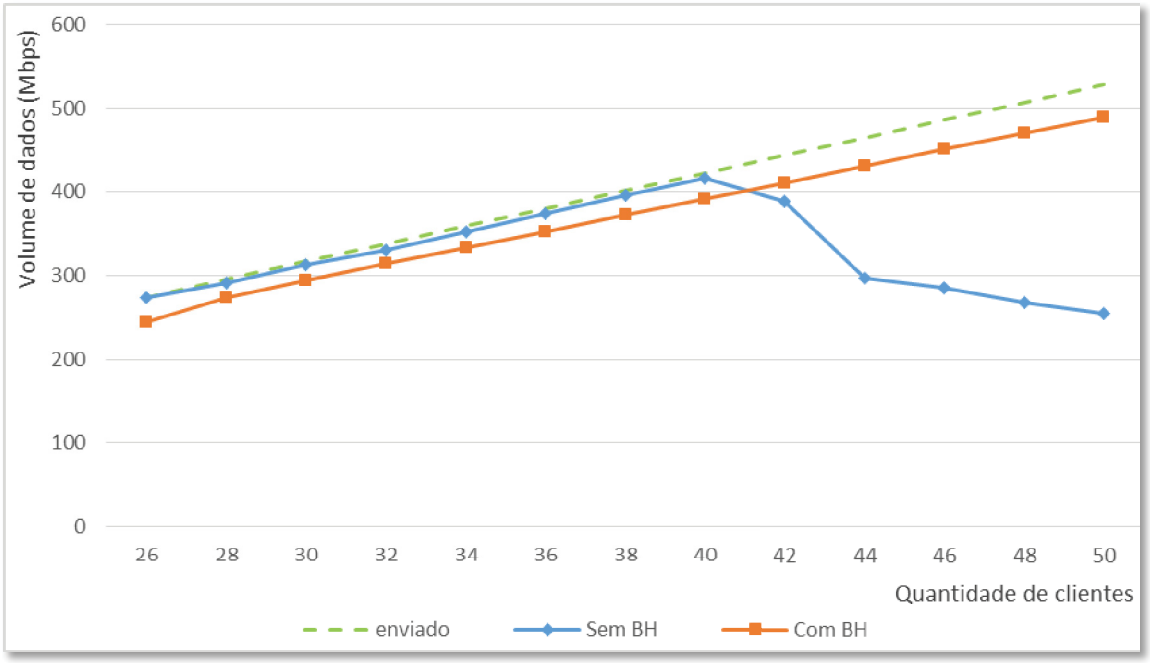
A Figura 19 apresenta o resultado médio dos testes executados com cada configuração de clientes IPEREF (com UDP), utilizada para o limite de 10 Mbps. Os eixos do gráfico foram ajustados para a escala de volume de Megabits por segundo, enviados para facilitar a visualização da diferença entre a utilização do BH e a não utilização. Nele é possível perceber que, utilizando o mecanismo, o volume de tráfego recebido acompanha o enviado, existindo uma pequena distância entre eles. Já sem utilizá-lo, o volume de tráfego recebido permanece abaixo de 180 Mbps e se assemelha a uma linha horizontal.

Figura 19 – Média de dados UDP recebidos conforme volume enviado, num limite de 10 Mbps na S-C, com e sem BH



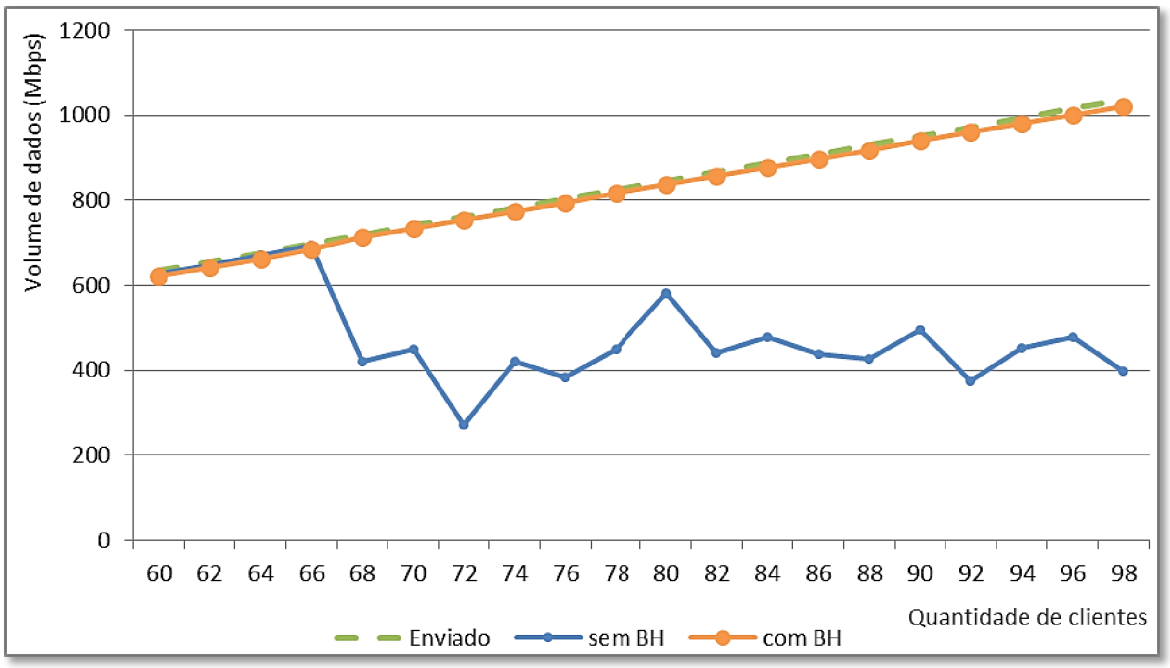
A Figura 20 apresenta o resultado médio dos testes executados com cada configuração de clientes IPEREF utilizada para o limite de 20 Mbps, na mesma formação da Figura 19. É possível perceber que, utilizando o mecanismo, o volume de tráfego recebido acompanha o enviado, existindo uma pequena distância entre eles. Já sem utilizar o BH, o volume de tráfego recebido permanece abaixo de 420 Mbps, que representa os testes com 40 clientes IPERF.

Figura 20 – Média de dados UDP recebidos conforme volume enviado, num limite de 20 Mbps na S-C, com e sem BH



No mesmo formato das figuras anteriores, foi gerada a Figura 21, agora para o limite de 40 Mbps, na qual é possível perceber que, utilizando o mecanismo, o volume de tráfego recebido também acompanha o enviado, existindo uma pequena distância entre eles. Já sem utilizar o BH, o volume de tráfego recebido permanece abaixo de 690 Mbps, que representa os testes com 66 clientes IPERF.

Figura 21 – Média de dados UDP recebidos conforme volume enviado, num limite de 40 Mbps na S-C, com e sem BH



É interessante comparar a Figura 22 e a Figura 23, pois elas representam os testes com volume de tráfego limiares anterior e posterior ao ponto de sobrecarga, respectivamente. Na primeira, o volume de dados recebido está adequado, mesmo sem o mecanismo. Na segunda, é possível observar a degradação do volume de tráfego logo após o primeiro HT. Utilizando o mecanismo de descarte, os momentos de HT são bem marcados com quedas no volume de tráfego, porém são pontuais.

Figura 22 – Volume de dados UDP recebidos pelo servidor IPERF por 66 clientes, com o limite de 40 Mbps na S-C

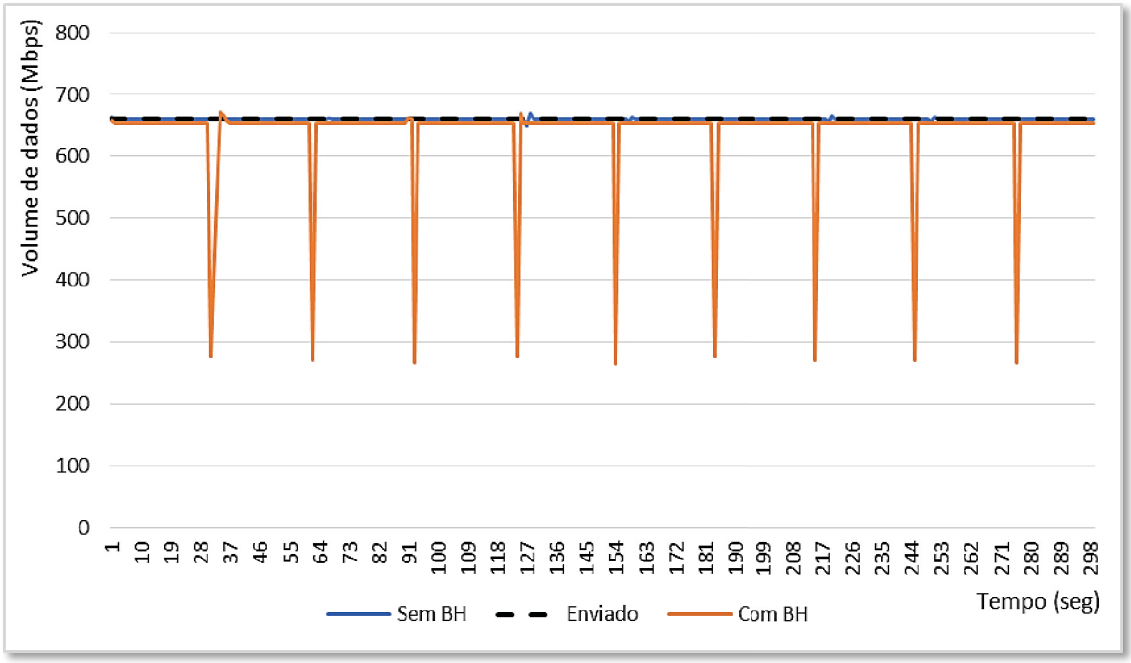
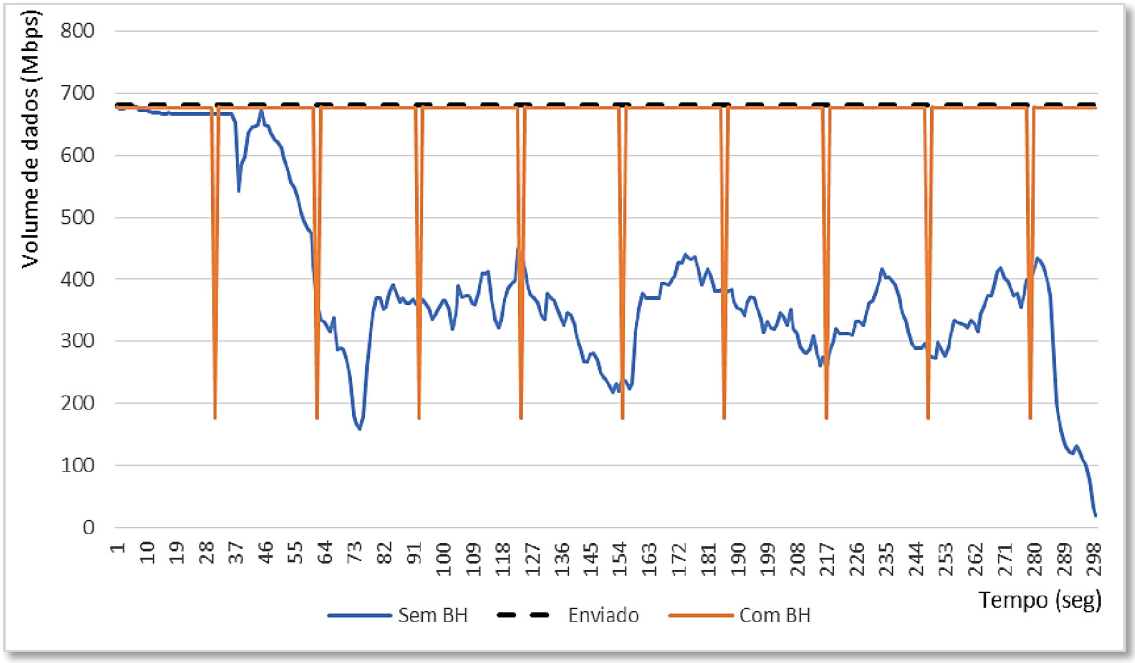


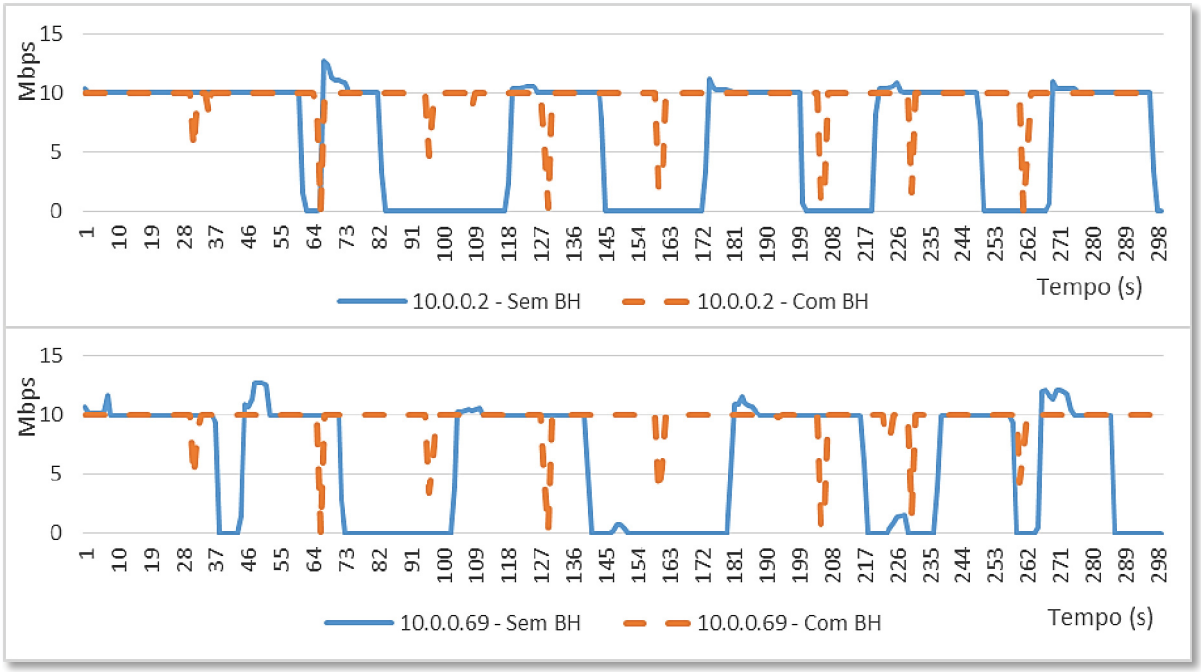
Figura 23 – Volume de dados UDP recebidos pelo servidor IPERF por 68 clientes, com o limite de 40 Mbps na S-C



O fenômeno ocorrido devido à sobrecarga pode também ser analisado pelo comportamento do volume de tráfego recebido para cada origem, conforme Figura 24. Nela, há os mesmos dados da Figura 23, mas em dois gráficos para uma transmissão cada (com e sem BH). Uma observação

interessante é que com BH os clientes dos dois gráficos têm o HT sincronizados, mas os outros dois não. Isso ocorre porque a configuração das rotas deixa de ser síncrona e imediata quando há sobrecarga da interface com controlador.

Figura 24 – Detalhamento da transferência de 4 origens de dados UDP para um dos testes de 68 clientes IPERF, com limite de 40 Mbps na S-C



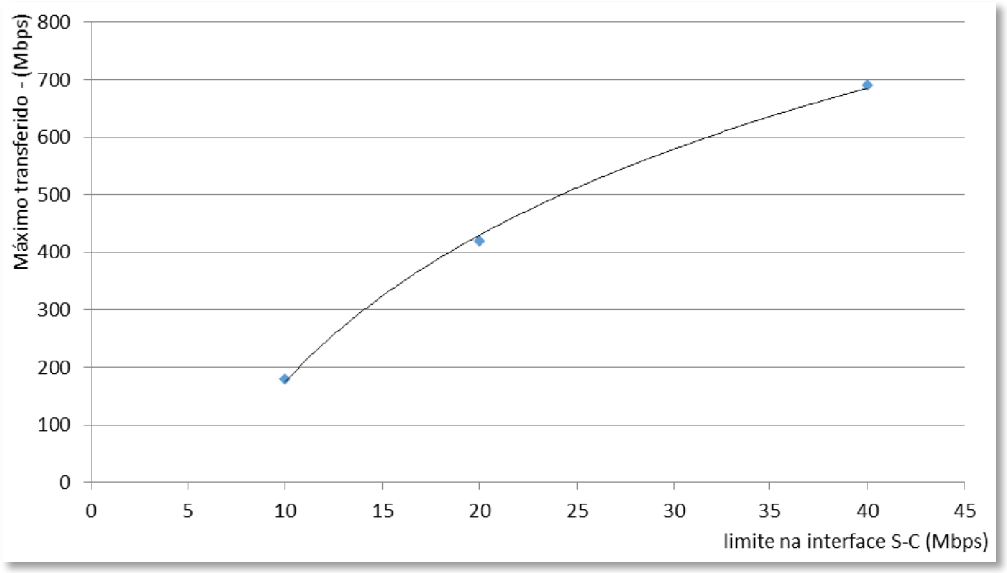
Nos testes foi confirmada a limitação de desempenho do Mininet, pois não foi possível realizar testes com mais de 98 transmissões simultâneas. Ele chega a criar os terminais e suas conexões, porém, não reconhece todos os comandos de envio de pacotes e impossibilita a avaliação de congestionamento da S-C. Nos testes com mais de 80 transmissões, houve a necessidade de descartar execuções (e refazê-las) devido a essa limitação. É possível que em um ambiente com maior capacidade de processamento ou em uma nova versão do software o Mininet apresente um desempenho melhor.

Assim ficou inviável avaliar os resultados com limites maiores na S-C, afinal, seria necessária maior quantidade de clientes IPERF transmitindo simultaneamente. Essa limitação impactou também os testes com transmissões TCP, pois, com uma tolerância maior, são necessárias mais transmissões concorrentes para identificar o limite do plano de dados. Por outro lado, uma maior tolerância implica em menor necessidade de explorar a relação entre a capacidade da S-C com a demanda de tráfego.

Em todos os testes de transmissão UDP, o recebimento do tráfego teve o mesmo comportamento. Com o BH, o volume foi pouco menor que o do enviado independentemente da quantidade de geradores de tráfego. Sem o mecanismo, o comportamento apresentado é que, após um limite, o volume de tráfego recebido é significativamente menor (20% ou mais) que o volume enviado. Conforme aumenta o volume de tráfego, existe a tendência de o volume recebido permanecer igual ou, ainda, reduzir.

O BH teve sucesso em reduzir o consumo de banda para controle para todas as velocidades e quantidades de transmissões com UDP. É possível criar uma relação entre os resultados apresentados nas figuras 19, 20 e 21 e, assim, ter uma linha de tendência para a sobrecarga do plano de dados em relação à capacidade da S-C, conforme figura 25. Foi observado em algumas simulações um alto volume de pacotes descartados, que parece estar relacionado com algum problema na implementação do OVS e que merece ser investigado futuramente. Quando corrigido, haverá redução significativa na quantidade de pacotes descartados.

Figura 25 – Relação entre a capacidade do plano de dados e a da S-C para UDP nos testes realizados



Os testes com maior quantidade de terminais com TCP tiveram como característica a redução gradativa do volume de dados transmitidos conforme aumentava a demanda. Utilizando ou não o BH, o desempenho era semelhante, a exemplo da transmissão de 22 clientes num limite de 10 Mbps na S-C,

conforme Figura 26. Com o BH, o protocolo também realiza ajustes na janela de transmissão e a linha verde da figura mostra essa variação.

Conforme a Figura 27, nos testes com limite de 10 Mbps na S-C. O volume de dados recebidos ficava mais longe da capacidade da interface T-S conforme aumentava a quantidade de clientes IPERF. A variação surgiu a partir da transmissão de 18 origens, mesmo momento em que começou a haver perda utilizando UDP.

Figura 26 – Média de dados TCP recebidos pelo servidor IPERF por 22 clientes com e sem BH e a média da janela com BH, para o limite de 10 Mbps na S-C

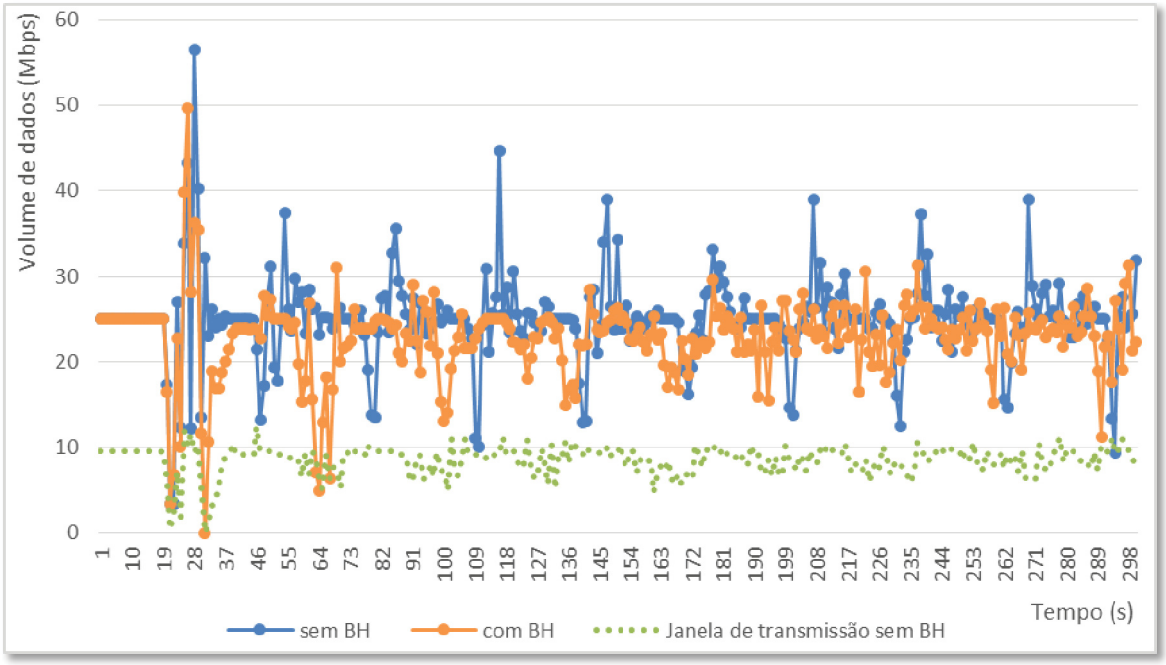
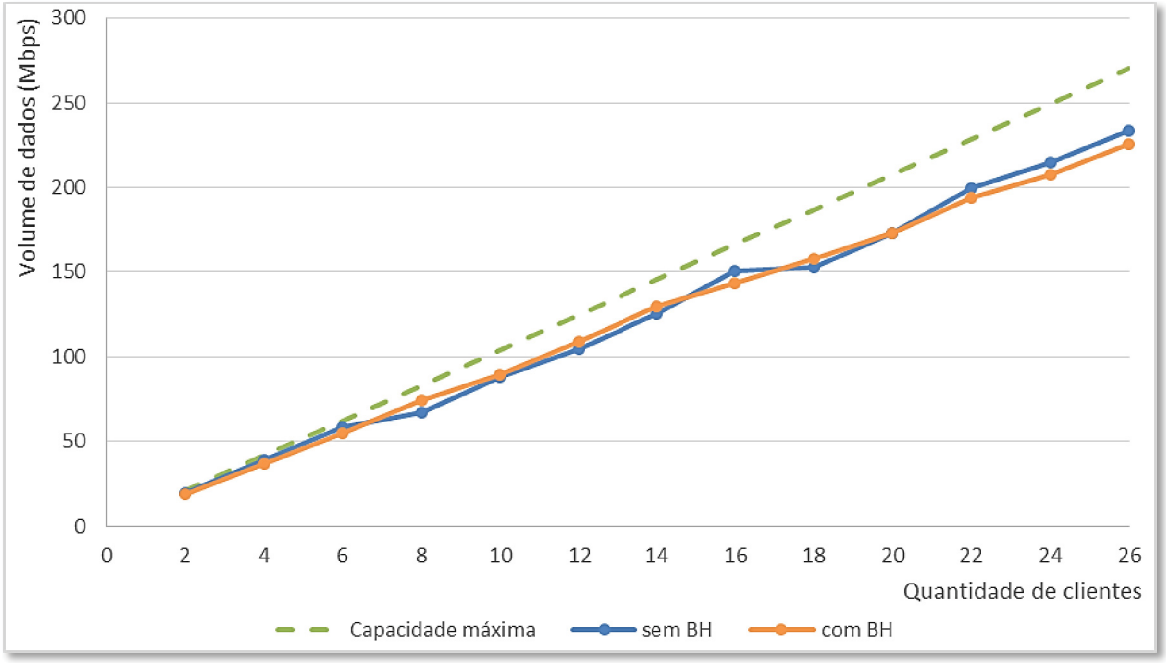


Figura 27 – Média de dados TCP recebidos conforme volume enviado num limite de 10 Mbps na S-C, com e sem BH



5 CONCLUSÃO

Este trabalho apresentou os resultados de testes em SDN, que permitiram medir o volume de pacotes enviados para o controlador em condições de sobrecarga e avaliar uma solução mitigadora baseada em descarte de pacotes. A partir dos resultados, foi confirmado que o congestionamento na interface impacta os serviços do controlador e, assim, afeta a qualidade do serviço destinado ao usuário da rede. Houve impacto na qualidade do serviço transportado por *User Datagram Protocol* (UDP) por enviar pacotes fora de ordem e por deixar de enviar pacotes. Para *Transmission Control Protocol* (TCP) houve apenas redução do volume de dados transmitidos.

Várias simulações permitiram medir a banda mínima entre controlador e switch para que o serviço não seja afetado pela falta de mensagens de controle durante o estabelecimento de um fluxo na topologia estudada. O resultado foi 8% da banda do plano de dados para UDP e 2% para TCP. Estes valores podem ajudar no dimensionamento de controles e ou na implementação de mecanismo adaptativo de descarte. O mecanismo Blackhole poderia ser ativado por um gatilho no controlador que mede o volume de pacotes recebidos ou pela ocupação da interface com o switch e, então, enviaria um comando.

O mecanismo de descarte apresentou-se como uma solução melhor que o comportamento padrão para transmissões UDP em Software Defined Network (SDN) para situações de sobrecarga do controlador. Mesmo devido ao descarte, o mecanismo garantiu um volume menor de perda do que o verificado sem o mecanismo de descarte. Assim, no planejamento de novas redes, poderão ser utilizadas informações deste estudo para implementar está solução de descarte para proteção da comunicação com o controlador e prevenção da degradação de serviço no plano de dados. Para TCP não foram verificadas vantagens para o serviço, pois observou-se a redução do tráfego devido a atuação do controle de congestionamento do protocolo de transporte.

Mais testes podem contribuir para avaliar outros cenários. Pode-se estudar, por exemplo, a utilização de outras topologias, outros controladores, outros protocolos de transporte, outros tipos de dados, outras taxas de transferência e outros emuladores. Outros trabalhos relacionados a esse tema

apresentam soluções que podem ser utilizadas para mitigar o problema e podem ser validados, como o FFS no controlador.

REFERÊNCIAS

AKAMAI. **State of the internet, Q4 2016 report**, v. 9, n. 4, 2016. Disponível em: < https://www.akamai.com/kr/ko/multimedia/documents/state-of-the-internet/q4-2016-state-of-the-internet-connectivity-report.pdf >. Acesso em: 1 Jul. 2018.

BADDELEY, M.; NEJABATI, R.; OIKONOMOU, G.; GORMUS, S.; SOORIYABANDARA, M.; SIMEONIDOU, D. **Isolating SDN control traffic with layer-2 slicing in 6TiSCH industrial IoT networks**. IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). Berlin, pp. 247-251, 2017.

CISCO. **Cisco visual networking index**: Forecast and methodology, White paper, p. 15, Mai. 2016.

FAVARO, A.; RIBEIRO, E. P. **Reducing SDN/openflow control plane overhead with blackhole mechanism**. GLOBAL INFORMATION INFRASTRUCTURE AND NETWORKING SYMPOSIUM (GIIS), 2015. Guadalajara, pp. 1–4, Out. 2015.

HOMBASHI, Tsuyoshi, **tcconfig**, 2018. [On-line]. Disponível em: <https://readthedocs.org/projects/tcconfig/>. Acesso em: 1 jul. 2018.

GUEDES, D. *et al*. **Redes definidas por software: uma abordagem sistêmica para o desenvolvimento das pesquisas em redes de computadores**. XXX SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS. Ouro Preto, 2012. Disponível em: <http://homepages.dcc.ufmg.br/~mmvieira/cc/papers/minicurso-sdn.pdf>. Acesso em 1 jul. 2018.

KREUTZ , D. *et al*. **Software-Defined Networking: A Comprehensive Survey**. Proceedings of the IEEE, v. 103, pp. 14-76, Out. 2014.

KIM, S.-S. *et al*. **A cognitive model-based approach for autonomic fault management in openflow networks**. Int. Journal of Network Management, v. 23, n. 6, pp. 383–401, 2013.

MCCAULEY, Murphy. **POX Manual Current documentation**, 2015. Disponível em: <https://noxrepo.github.io/pox-doc/html/>. Acesso em: 1 jul. 2018.

MCMAHON, Robert. **Iperf2**, Maintenance and enhancements of iperf 2.0.5, 2018. Disponível em: <https://sourceforge.net/projects/iperf2/>. Acesso em: 1 jul. 2018.

MENDOZA, F.; FERRÚS, R.; SALLENT, O. **SDN-based Traffic Engineering for Improved Resilience in Integrated Satellite-Terrestrial Backhaul Networks**. INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGIES FOR DISASTER MANAGEMENT (ICT-DM). Münster, pp. 1-8, 2017.

MININET. **Mininet**: An Instant Virtual Network on your Laptop (or other PC), 2018. Disponível em: <http://mininet.org/>. Acesso em: 1 jul. 2018.

MOGUL, J. C *et al*. **Devoflow: cost-effective flow management for high performance enterprise networks**. 9TH ACM WORKSHOP ON HOT TOPICS IN NETWORKS. HOTNETS. Monterey, CA, USA, p. 1, Out. 2010.

MURAGAA, W. H.; SEMAN, K.; MARHUSIN, M. F. **A pox controller module to collect web traffic statistics in SDN environment**. World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, v. 10, n. 12, pp. 2002–2007, 2016.

ONF. **Software-defined networking**: The new norm for networks, ONF White Paper, 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdnresources/white-papers/wp-sdn-newnorm.pdf>. Acesso em: 1 jul. 2018.

OPENVSWITCH. **Open vSwitch Manual – ovs-ofctl (8)**, Open vSwitch Manpages, 2018. Disponível em: <http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt>. Acesso em: 1 jul. 2018.

PAVLOU, G. **On the evolution of management approaches, frameworks and protocols: A historical perspective**. J. Netw. Syst. Manage., v. 15, n. 4, pp. 425–445, Dez. 2007.

POLAT, H.; POLAT, O. **The effects of DoS attacks on ODL and POX SDN controllers**. 8th International Conference on Information Technology (ICIT). Amman, pp. 554-558, 2017.

RAMOS, R. M.; MARTINELLO, M.; ROTHENBERG, C. E. **Data center fault-tolerant routing and forwarding: An approach based on encoded paths**. SIXTH LATIN-AMERICAN SYMPOSIUM ON DEPENDABLE COMPUTING (LADC). Rio de Janeiro, pp. 104–113, 2013.

RODRÍGUEZ, F. L. **Arquitetura e protótipo de uma rede sdn-openflow para provedor de serviço**. 2014. 72 f. Dissertação (Mestrado) – Faculdade de Tecnologia, Universidade de Brasília, Brasília, 2014.

SHENKER, S. **The future of networking, the past of protocols**, 2011. Disponível em: <http://www.youtube.com/watch?v=YHeyuD89n1Y>. Acesso em: 1 jul. 2018.

SHERRY, J. *et al*. **Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service**. Computer Communication Review, 4 ed., v. 42, p. 13-24, 2012.

TANENBAUM, A. S. **Redes de computadores**. Rio de Janeiro: Campus, 1997.

TATANG, D.; QUINKERT, F.; FRANK, J.; ROPKE, C.; HOLZ, T. **SDN-Guard: Protecting SDN controllers against SDN rootkits**. IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). Berlin, pp. 297-302, 2017.

THAENCHAIKUN, C.; PANICHPATTANAKUL, W. **Control-plane OpenFlow segment routing SDN for network control messages reduction**. 14th International Conference on Electrical Engineering/Electronics, Computer,

Telecommunications and Information Technology (ECTI-CON). Phuket, pp. 322-325, 2017.

WANG, S. **Comparison of SDN OpenFlow Network Simulator and Emulators: EstiNet vs. Mininet**. SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC). Funchal: IEEE, 2014.

WIRESHARK FOUNDATION. **About Wireshark**, 2018. Disponível em: <https://www.wireshark.org/>. Acesso em: 1 jul. 2018.

APÊNDICE 1 – Código Python do controlador POX

```
# Copyright 2011-2012 James McCauley
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at:
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""
An L2 learning switch.

It is derived from one written live for an SDN crash course.
It is somwhat similar to NOX's pyswitch in that it installs
exact-match rules for each flow.
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time

log = core.getLogger()

# We don't want to flood immediately when a switch connects.
# Can be overriden on commandline.
_flood_delay = 0

class LearningSwitch (object):
  """
  The learning switch "brain" associated with a single OpenFlow switch.

  When we see a packet, we'd like to output it on a port which will
  eventually lead to the destination.  To accomplish this, we build a
  table that maps addresses to ports.

  We populate the table by observing traffic.  When we see a packet
  from some source coming from some port, we know that source is out
  that port.

  When we want to forward traffic, we look up the desintation in our
  table.  If we don't know the port, we simply send the message out
  all ports except the one it came in on.  (In the presence of loops,
  this is bad!).

  In short, our algorithm looks like this:

  For each packet from the switch:
  1) Use source address and switch port to update address/port table
  2) Is transparent = False and either Ethertype is LLDP or the packet's
     destination address is a Bridge Filtered address?
     Yes:
        2a) Drop packet -- don't forward link-local traffic (LLDP, 802.1x)
            DONE
  3) Is destination multicast?
     Yes:
        3a) Flood the packet
            DONE
  4) Port for destination address in our address/port table?
     No:
        4a) Flood the packet
            DONE
  5) Is output port the same as input port?
     Yes:
        5a) Drop packet and similar ones for a while
  6) Install flow table entry in the switch so that this
```

```
   flow goes out the appopriate port
   6a) Send the packet out appropriate port
"""
def __init__ (self, connection, transparent):
  # Switch we'll be adding L2 learning switch capabilities to
  self.connection = connection
  self.transparent = transparent

  # Our table
  self.macToPort = {}

  # We want to hear PacketIn messages, so we listen
  # to the connection
  connection.addListeners(self)

  # We just use this to know when to log a helpful message
  self.hold_down_expired = _flood_delay == 0

  #log.debug("Initializing LearningSwitch, transparent=%s",
  #          str(self.transparent))

def _handle_PacketIn (self, event):
  """
  Handle packet in messages from the switch to implement above algorithm.
  """

  packet = event.parsed

  def flood (message = None):
    """ Floods the packet """
    msg = of.ofp_packet_out()
    if time.time() - self.connection.connect_time >= _flood_delay:
      # Only flood if we've been connected for a little while...

      if self.hold_down_expired is False:
        # Oh yes it is!
        self.hold_down_expired = True
        log.info("%s: Flood hold-down expired -- flooding",
            dpid_to_str(event.dpid))

      if message is not None: log.debug(message)
      #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
      # OFPP_FLOOD is optional; on some switches you may need to change
      # this to OFPP_ALL.
      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    else:
      pass
      #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
    msg.data = event.ofp
    msg.in_port = event.port
    self.connection.send(msg)

  def drop (duration = None):
    """
    Drops this packet and optionally installs a flow to continue
    dropping similar ones for a while
    """
    if duration is not None:
      if not isinstance(duration, tuple):
        duration = (duration,duration)
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet)
      msg.idle_timeout = duration[0]
      msg.hard_timeout = duration[1]
      msg.buffer_id = event.ofp.buffer_id
      self.connection.send(msg)
    elif event.ofp.buffer_id is not None:
      msg = of.ofp_packet_out()
      msg.buffer_id = event.ofp.buffer_id
      msg.in_port = event.port
      self.connection.send(msg)

  self.macToPort[packet.src] = event.port # 1

  if not self.transparent: # 2
    if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
```

```
        drop() # 2a
        return

    if packet.dst.is_multicast:
      flood() # 3a
    else:
      if packet.dst not in self.macToPort: # 4
        flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
      else:
        port = self.macToPort[packet.dst]
        if port == event.port: # 5
          # 5a
          log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
              % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
          drop(10)
          return
        # 6
        log.debug("installing flow for %s.%i -> %s.%i" %
                  (packet.src, event.port, packet.dst, port))
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.idle_timeout = 10
        msg.hard_timeout = 30 #Alterado.
        msg.actions.append(of.ofp_action_output(port = port))
        msg.data = event.ofp # 6a
        self.connection.send(msg)

class l2_learning (object):
  """
  Waits for OpenFlow switches to connect and makes them learning switches.
  """
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)

def launch (transparent=False, hold_down=_flood_delay):
  """
  Starts an L2 learning switch.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  core.registerNew(l2_learning, str_to_bool(transparent))
```

APÊNDICE 2 – Código Python para automatização dos testes UDP

```
#!/usr/bon/env python
#Utilização:
#1º Iniciar controlador (ex: POX) e realizar outras configurações.
#2º realizar o seguinte comando: $sudo python script.py [aplicar Blackhole (Y) ou não (N)] [Repetições do teste] [Hosts
totais, origens + 1 destino] [Banda de transmissão em Mbps]
#Ex: sudo python Auto_varias_origens_IPERF.py Y 3 21 10
#Criado em 2017 por Conrado Bozza
import sys #argumentos
import subprocess #para abrir programa do mininet
import time #para executar as pausas

BH=str(sys.argv[1]) #Aplicar blackhole (Y) ou nao (N).
R=int(sys.argv[2]) #Quantidade de repeticoes do teste (1,2,3,...).
Hosts=int(sys.argv[3]) #Quantidade de Hosts do teste (1,2,3,...).
banda=int(sys.argv[4]) #Banda de transmissao dos IPERFs em Mbps.
x=1
topo = 'single,' + str(Hosts)

print 'start'
for x in range(R):
   xx=2
   #p = subprocess.Popen(['sudo', 'mn','--switch','ovsk,protocol=OpenFlow13', '--controller','remote', '--topo','single,2'],
stdin=subprocess.PIPE, stdout=subprocess.PIPE)
   p = subprocess.Popen(['sudo', 'mn','--switch','ovsk,protocol=OpenFlow10', '--controller','remote', '--topo',topo],
stdin=subprocess.PIPE, stdout=subprocess.PIPE)
   time.sleep(6)
   p.stdin.write('s1 ovs-ofctl del-flows "s1"')
   if (BH == "Y"):
      p.stdin.write('s1 ovs-ofctl add-flow "s1" "table=0 priority=0 actions=controller, learn(table=0,
priority=100,hard_timeout=1,NXM_OF_ETH_DST[],NXM_OF_ETH_SRC[],NXM_OF_ETH_TYPE[],)"\n')
   time.sleep(4)
   p.stdin.write('s1 ovs-ofctl dump-flows "s1"\n')
   time.sleep(4)
   p.stdin.write("h1 iperf -s -u -p 5001 -i 1 -y C > /home/BH/IPERF_%dh_%dMbps_%s_BH_%d.csv &\n"
%(Hosts,banda,BH,x))
   time.sleep(4)
   for xx in range(2,Hosts+1):
      p.stdin.write(" h%d iperf -c h1 -u -p 5001 -i 1 -t 300 -b %dM/sec &\n" %(xx,banda))
      time.sleep(0.0001)
   #print "Comandos do teste %d enviados..." % x
   time.sleep(310)
   p.stdin.write("exit\n")
   time.sleep(6)

stdout, stderr = p.communicate()
print stdout

print 'end'
```