UNIVERSIDADE D
COIMBRA

Domitília Maria Bernardes Noro

SEGURANÇA PARA AMBIENTES SDN COM P4

Dissertação no âmbito do Mestrado em Segurança Informática, orientada pelos Professores Doutores Paulo Simões e Tiago Cruz e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Janeiro de 2024

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Domitília Maria Bernardes Noro

SEGURANÇA PARA AMBIENTES SDN COM P4

Dissertação no âmbito do Mestrado em Segurança Informática, orientada pelos Professores Doutores Paulo Simões e Tiago Cruz e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Janeiro de 2024

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA
DEPARTMENT OF INFORMATICS ENGINEERING

Domitília Maria Bernardes Noro

SECURITY FOR SDN ENVIRONMENTS WITH P4

Dissertation in the scope of the Master's in Informatics Security, advised by Prof. Paulo Simões and Prof. Tiago Cruz and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

January 2024

Agradecimentos

Gostaria de agradecer aos meus orientadores, Professor Doutor Paulo Simões e Professor Doutor Tiago Cruz, pela orientação consistente, apoio e paciência ao longo deste processo.

Refletindo sobre o caminho percorrido, é inevitável reconhecer a importância da colaboração com os meus colegas de mestrado Duarte Dias e Rui Pires. Aprecio a sua notável colaboração e espírito de equipa, que muito contribuíram para a superação de desafios e enriquecimento do resultado final deste trabalho.

Expresso ainda, a minha profunda gratidão à minha família, amigos e colegas, mas muito especialmente ao meu marido e filha, cujo apoio constante, motivação contínua e crença em mim, foram a força que me levou a superar obstáculos e a persistir, mesmo nos momentos mais difíceis. Compreendendo as exigências deste projeto, eles ofereceram um apoio incondicional, abrindo mão do meu tempo com eles, para que eu pudesse dedicar-me a ele.

Este trabalho não seria possível sem a contribuição de cada um de vocês. Agradeço sinceramente a todos que, de uma forma ou de outra, contribuiu para tornar este percurso possível.

Resumo

A Internet tem um papel fundamental para a formação da sociedade digital atual, cuja dependência de dispositivos interligados e contactáveis a partir de qualquer lugar, cresce significativamente a cada dia. Com a adoção massificada destes dispositivos veio a necessidade das redes suportarem uma gama maior de funcionalidades de comunicação, resultando em sistemas cada vez mais complexos e difíceis de gerir. Isto implica que as organizações necessitam de padrões de computação mais avançados, para fazer face ao desafio das necessidades emergentes. Consequentemente, isto requer mais e mais recursos, não só do ponto de vista computacional, mas também de planeamento, gestão, disponibilidade e escalabilidade, para as quais, as tecnologias tradicionais já não conseguem dar resposta à complexidade necessária. Num esforço de dar resposta a estas novas exigências, comunidades de código aberto perceberam que era necessária uma nova filosofia de design em rede. Nesse esforço foi proposto o paradigma de Redes Definidas por S*oftware* (SDN), uma abordagem de rede que utiliza redes programáveis como solução de conectividade. A nova programabilidade das redes traz flexibilidade e acelera a implementação de novas soluções, mas traz consigo novos desafios, nomeadamente do ponto de vista da segurança.

Este trabalho analisa as preocupações gerais de segurança em SDN. Primeiramente é apresentada uma visão geral da SDN, da sua arquitetura e capacidades. Seguidamente, é apresentada uma análise da segurança na SDN onde são abordadas as suas vulnerabilidades, ataques, e estratégias de mitigação, com especial ênfase no plano de dados. Depois faz-se uma análise dos diferentes ataques de negação de serviço distribuídos (DDoS - *Distributed Denial of Service*) nas redes convencionais, bem como nas redes baseadas no paradigma SDN. São também revistos os mecanismos de defesa associados a estes, implementados no contexto de SDN. Adicionalmente são revistos conceitos de *OpenFlow*, P4, arquitetura PISA e os dispositivos de rede programáveis e identificadas as vantagens do plano de dados programável baseado em P4

Seguidamente, é apresentada a metodologia utilizada neste trabalho, juntamente com a linha do tempo que ilustra a distribuição das tarefas associadas ao longo do período de conclusão previsto.

Finalmente, é apresentada a solução DPSynFloodBlock, desenvolvida utilizando a linguagem P4, com o propósito de monitorar, detetar, mitigar e prevenir ataques de DDoS *SYN Flood*, viabilizando a realização dessas ações sem a necessidade de intervenção do controlador. Nesta solução, embora seja possível bloquear os ataques por meio das diretrizes provenientes do controlador após a transmissão de informações pelos dispositivos programáveis, o seu foco principal é conferir capacidade decisória ao plano de dados sem depender da intervenção direta do controlador. Isso é alcançado por meio da utilização exclusiva das informações provenientes dos cabeçalhos dos fluxos de rede e dos dados inicialmente fornecidos pelo controlador durante a inicialização do dispositivo programável, utilizando as tabelas de correspondência de ação.

Palavras-Chave

SDN, plano de dados, dispositivos de rede programáveis, P4, DDoS.

Abstract

The Internet has played a key role in shaping the today's digital society, for which there is an increasing dependence on interconnected devices that can be accessed from anywhere. With its mass adoption of the use of these devices came the need to increase the networks that support their communication, which become increasingly complex. This means that organizations need to use more advanced computing patterns to meet the challenge of emerging needs. Therefore, it requires more and more resources, not only from the point of view of computation, but also from planning, management, availability, and scalability. Moreover, traditional technologies are no longer able to respond to the complexity required. To meet these new demands, open-source communities realized that a new network design philosophy was needed. In this effort, the *software*-defined network (SDN) was proposed, a networking solution through programmable networks.

Network programmability brings flexibility and accelerates the implementation of new solutions, but also, brings with it new challenges, particularly from the perspective of security.

This work examines general security concerns in SDN. Firstly, an overview of SDN, its architecture, and capabilities is presented. Next, a security analysis in SDN is provided, addressing its vulnerabilities, attacks, and mitigation strategies, with a special focus on the data plane. An analysis of various Distributed Denial of Service (DDoS) attacks in both conventional and SDN-based networks is conducted. The defense mechanisms implemented in the SDN context against these attacks are also reviewed. Additionally, concepts of OpenFlow, P4, PISA architecture, and programmable network devices are discussed, highlighting the advantages of the programmable data plane based on P4.

Next, the methodology used in this work is presented, along with the timeline illustrating the distribution of associated tasks throughout the anticipated completion period.

Finally, the DPSynFloodBlock solution developed using the P4 language is presented, aimed at monitoring, detecting, mitigating, and preventing *SYN Flood* DDoS attacks, enabling the execution of these actions without the need for controller intervention. In this solution, although it's possible to block attacks through instructions provided by the controller - after it received information by the programmable devices - its primary focus is to empower the decision-making capability of the data plane without relying on direct controller intervention. This is achieved exclusively through information derived from network flow data and network related information provided by the controller during the programmable device initialization, through action matching tables.

Keywords

SDN, data plane, programmable network devices, P4, DDoS.

Índice

Acrónimos

ACL	Access Control List
AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
API	Application Programming Interface
ASIC	Application-specific Integrated Circuit
CLI	Command Line Interface
COTS	Commercially Off-The-Shelf
CP	Control Plane
CPU	Central processing Unit
CSRF	Cross Site Request Forgery
DDoS	Distributed Denial of Service
DoS	Denial of Service
DP	Data Plane
DPI	Deep Packet Inspection
DRDoS	Distributed Reflection Denial of Service
FPGA	Field-Programmable Gate Arrays
IDS	Intrusion Detection System
IoT	Internet-of-Things
IPS	Intrusion Protection System
LLDP	Link Layer Discovery Protocol
MAT	Match Action Table
MITM	Man-In-The-Middle
NBI	Northbound Interface
NIC	Network Interface Card
NPU	Network Processing Unit
ONF	Open Networking Foundation
QoS	Quality of Service
P4	Programming Protocol-Independent Packet Processors
PISA	Protocol-Independent Switch Architecture
PSA	Portable Switch Architecture
REST	Representational State Transfer
RTO	Retransmission Time Out
RTT	Retransmission Trip-Time
SBI	Southband Interface
SDN	Software Defined Network
SNMP	Simple Network Management Protocol.
SRAM	Static Random-Access Memory
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TNA	Tofino Native Architecture
VM	Virtual Machine

XSS Cross Site Scripting
XXHASH Extremely Fast Hash Algorithms

Lista de Figuras

Lista de Tabelas

Capítulo 1
Introdução

A introdução da Internet permitiu um desenvolvimento extraordinário nas comunicações levando a um desenvolvimento sem par das tecnologias até aí existentes e o aparecimento de muitas outras que permitiram o avanço da humanidade a uma escala sem precedentes. No entanto todo este desenvolvimento trouxe uma pressão para as redes tradicionais, que devido ao número de sistemas, serviços, utilizadores e consequente necessidade de equipamentos de rede mais sofisticados e eficientes, acaba por condicionar a introdução de novas tecnologias. Esta necessidade de um sistema de comunicação mais escalável, fiável, com custos reduzidos que permita uma gestão ágil e flexível do ponto de vista de gestão da mesma, levou ao aparecimento de Redes definidas por *Software* (SDN – *Software Defined Networking*), inicialmente proposta pela Universidade de Stanford. e posteriormente desenvolvida e padronizada pelo consórcio ONF (*Open Networking Foundation*).

Através de um design onde é feita a separação das funções de dados (o sistema que encaminha os pacotes para o destino) e de controlo (o sistema que toma decisões) a SDN permite uma maior abstração e flexibilidade relativa aos equipamentos de rede. A limitação da implementação e desenvolvimento de novos protocolos por parte das redes tradicionais, caracterizadas por terem designs proprietários baseados em implementações codificadas em chips, fez com que os sistemas SDN, que são menos complexos, permitissem um desenvolvimento mais rápido e mais baratos de novas soluções de comunicação.

O facto de centralizar as funções de controlo da rede num equipamento dedicado unicamente para esta responsabilidade, em vez de instalado em todos os dispositivos de rede, permitiu que estes tivessem requisitos de CPU não tão elevados e consequentemente os custos diminuíssem e a inovação acelerasse.

Com o controlador a tomar decisões baseados na análise dos fluxos de tráfego e de estatísticas obtidas dos pacotes que atravessam a rede, passa a ser possível detetar e reagir a alterações ou anomalias da rede de forma centralizada [1].

Similarmente a quase todas as novas tecnologias, também a arquitetura SDN introduz novos vetores de ataque e com eles novos desafios do ponto de vista da segurança. Apesar dos benefícios introduzidos pelo controlador comum, onde são decididas as ações do *hardware* de rede, a centralização é mais suscetível a erros, a uso malicioso e à possibilidade de sobrecarga do único ponto encarregue de supervisionar operações quer de alto, quer de baixo nível.

Este estrangulamento aliado à função fixa do plano de dados (DP) bem como uma heterogeneidade das soluções proprietárias dos dispositivos de rede levou à necessidade de desenvolvimento de soluções que permitissem programar os equipamentos de rede, delegando tarefas de baixo nível do controlador de SDN para a camada de infraestrutura [2]. Este novo melhoramento só foi conseguido com o desenvolvimento de linguagens de programação no DP como seja a linguagem P4.

O P4 (*Programming Protocol-independent Packet Processors*) é uma linguagem de programação específica para dispositivos de rede, que descreve de forma agnóstica como os dispositivos do DP processam os pacotes [3] permitindo uma abstração total do *hardware* subjacente.

No entanto, esta evolução veio também adicionar mais vetores de ataque de rede do DP, aos já introduzidos devido ao desacoplamento do plano de controlo (CP) e do DP. A exploração da característica de programação do comportamento de encaminhamento pode ter repercussões enormes no funcionamento de toda a rede.

Apesar de ao longo dos últimos anos terem sido feitos consideráveis esforços para melhorar a segurança do plano de controlo de SDN, devido ao impacto que este tem na segurança de toda a rede, consequência da sua centralidade, foram menos aqueles dedicados a encontrar soluções de mitigação de vulnerabilidades do DP, não obstante as consequências similares em termos de impacto na rede [4]. Por um lado, devido às limitações das soluções anteriores ao uso de P4, e por outro devido à programação total deste ser relativamente recente. Ainda que os equipamentos de rede programáveis sejam limitados em termos da computação permitida a cada pacote, tipos de operações suportadas ou memória, o facto de o processamento de pacotes no plano de dados ser feito a uma velocidade quase instantânea faz com que as soluções passíveis de serem desenvolvidas neste plano possam impedir/mitigar ataques mais rapidamente [5]. Numa altura em que ataques maliciosos têm crescido enormemente e com efeitos cada vez maiores, como sejam os ataques de DDoS (*Distributed Denial of Service*), quaisquer ganhos em tempo de resposta que permitam reduzir o tempo em que a rede está vulnerável, assim como a deteção do ataque no ponto de entrada mais perto do atacante, ou ainda a rápida reconfiguração dos equipamentos de rede, limitará as consequências no funcionamento da rede.

Nesse sentido, o desenvolvimento de soluções dinâmicas, com informação de estados ou *stateful* no DP permitirá às redes de SDN tornarem-se mais eficientes, flexíveis, escaláveis garantindo comunicações onde as propriedades de segurança (confidencialidade, integridade e disponibilidade) sejam asseguradas. Por este motivo é fundamental a identificação das ameaças e vulnerabilidades associadas a plano de dados *stateful* bem como a especificação de estratégias para a sua mitigação. Adicionalmente, a análise das soluções desenvolvidas permitirá ter uma visão de possíveis melhorias que devem ser implementadas para uma solução mais segura.

Com os ataques de DDoS sendo os mais severos e destrutivos nas redes, como é o exemplo do ataque sofrido pela Microsoft em novembro de 2021 com um *throughput* de 3.47 Tbps [6] é necessário investigar as possíveis estratégias de deteção, proteção e mitigação nas redes SDN e em particular as aplicáveis ao plano de dados. Estratégias estas, que poderão ajudar a resolver os problemas de outras soluções implementadas noutros planos, mas que têm um custo associado ao desempenho do processamento de pacotes e incorrem em latência ponto-a-ponto não trivial, o que é inaceitável para alguns serviços de rede sensíveis à latência. Assim, é importante encontrar soluções a ataques de DDoS a partir do processamento de pacotes em tempo real, nos pipelines dos dispositivos programáveis, potenciando medidas flexíveis, com alto desempenho e de baixo custo.

1.1 Objetivos Gerais

Este estudo tem como objetivo principal abordar questões de segurança de Redes Definidas por *Software* (SDN), com foco no plano de dados. Para isso pretende-se compreender a natureza da SDN, explorando a sua arquitetura, os seus benefícios, desafios e sobretudo os problemas de segurança associados a elas. Além disso, pretende-se investigar os benefícios decorrentes da implementação de soluções voltadas para a segurança no plano de dados, bem como compreender o funcionamento e as vantagens da programação utilizando a linguagem P4 nesse contexto. O estudo também visa analisar as atuais soluções de deteção e mitigação de ataques DDoS no âmbito da SDN. Sendo um dos principais objetivos avaliar a viabilidade de implementar uma solução focada na deteção e mitigação de ataques DDoS, através da programação em linguagem P4 nos dispositivos programáveis, utilizando exclusivamente informações provenientes dos fluxos de dados disponíveis no plano de dados.

1.2 Estrutura

O restante documento está organizado com a estrutura seguinte:

• Capítulo 2 – Refere os objetivos e o plano de trabalho implementado. Além disso, apresenta-se a estratégia usada para aquisição do conhecimento necessário e implementação do trabalho.

• Capítulo 3 - Apresenta uma visão geral dos tópicos-chave discutidos neste trabalho, nomeadamente sobre Redes Definidas por *Software*, onde são analisados os ataques e ameaças possíveis em ambientes SDN, bem como os desafios gerais deste paradigma. É ainda apresentado um estudo mais específico aos ataques DDoS onde são identificados quais os mecanismos de defesa que podem ser implementados em ambiente SDN para mitigação dos mesmos. Finalmente, é analisada a programabilidade do plano de dados, onde é feita uma análise detalhada à linguagem P4, e avaliadas as vantagens do plano de dados programável baseado em P4

• Capítulo 4 – Apresenta a solução desenvolvida em linguagem P4, DPSynFloodBlock, para dar resposta ao problema dos ataques de DDos *SYN Flood*, para programação dos dispositivos programáveis.

• Capítulo 5 – Apresenta a avaliação da solução DPSynFloodBlock. Para isso é feita uma comparação do seu comportamento com o comportamento de uma *SYN Flood*, onde são apresentados os resultados de algumas métricas de performance deste, nomeadamente de mémoria, de bandwidth e de RTT.

• Capítulo 6 - Apresenta as principais conclusões relativas ao trabalho elaborado, assim como o trabalho a desenvolver no futuro.

Capítulo 2
Objetivos e abordagem do estudo

Este capítulo descreve os objetivos principais deste estudo e a abordagem que foi feita no intuito de os alcançar. Inicialmente é descrita a metodologia usada, o plano de trabalho e os objetivos gerais do mesmo, bem como a motivação que levou até eles.

2.1 Metodologia

Neste trabalho efetuou-se uma revisão sistemática de literatura procurando material a partir de pesquisa nas plataformas *Google Scholar*, *IEEE Explore*, *Research gate*, *ScienceDirect*, *Google*, e *Academia.edu*. Nestas plataformas foram feitas pesquisas em inglês (para maior abrangência das fontes), usando as palavras-chave em inglês e combinações destas, nomeadamente: SDN, plano de dados, P4, segurança, ataques DoS, ataques DDoS e dispositivos de rede programáveis, onde foram identificados muitos artigos como se pode ver por alguns exemplos disponíveis na Tabela I, sendo impossível analisar todos.

Tabela I - Resultado da pesquisa de literatura nas plataformas Google Scholar, IEEE Explore, Research gate, ScienceDirect.

Pesquisa	Google Scholar	IEEE Explore	Research Gate	ScienceDirect
(software defined networking OR SDN) AND Data plane	20200		>34679	91424
(software defined networking OR SDN) AND security AND threats	23600	860	>37289	26257
(software defined networking OR SDN) AND security AND attacks	30700	2200	>28629	24456
(software defined networking OR SDN) AND Data plane AND security	17800	844	>27459	9352
(software defined networking OR SDN) AND P4	18400	326	>12529	8023
(software defined networking OR SDN) AND security AND P4	16500	121	>19299	1370
Software defined networking OR SDN	653000	24752	>16349	537,478
(software defined networking OR SDN) AND security AND attacks AND P4	8830	46	>18549	451
(software defined networking OR SDN) AND securityAND Data plane AND P4	17300	91	>20929	301
(software defined networking OR SDN) AND security AND P4 AND data plane AND programmable targets	17700	5	>13429	216
(software defined networking OR SDN) AND security AND attacks AND P4 AND data plane	6530	36	>16819	147
(software defined networking OR SDN) AND security AND attacks AND P4 AND data plane AND programmable targets	2250	2	>16899	122
(software defined networking OR SDN) AND Security	402000	5749	>24939	86,312
(software defined networking OR SDN) AND security AND attacks AND P4 AND data plane AND programmable targets AND DDoS	1790	160	>1000	63
(software defined networking OR SDN) AND security AND attacks AND P4 AND data plane AND programmable targets AND DoS	2000	105	>1000	58

Assim foi necessário definir critérios de exclusão e priorização. Optou-se por analisar as palavras-chaves individualmente e a combinação destas, com base nas datas de publicação. Como o conceito de SDN aparece pela primeira vez no final de 2009, considera-se só os artigos desde 2010 para pesquisas contendo esta palavra-chave.

Relativamente a pesquisas contendo a palavra-chave P4 são excluídos artigos anteriores a 2014 já que esta linguagem aparece pela primeira vez neste ano. Já para identificação de soluções de segurança em P4 para SDN só são considerados resultados desde 2018 pois a versão atual de P4, o P4-16 que só surgiu em 2016 pelo que se considerou que a falta de maturidade da linguagem entre 2016-2018 não levaria a resultados expressivos e, portanto, poderiam ser descartados. Em termos de priorização foi considerado

- inclusão do maior número de palavras-chave
- número de citações

- fontes publicadas em conferências ou revistas científicas.
- resultados repetidos em várias plataformas.

Após identificação e priorização do material a analisar, foi feita uma triagem através da leitura do sumário e conclusão dos mesmos, pesquisando os conteúdos enumerados acima na identificação do problema, a fim de tomar uma decisão sobre a leitura integral do artigo. Quando se decidiu por uma leitura integral de um artigo, foi feita uma análise das referências da mesma para verificar possível interesse destas, e verificadas se as mesmas já haviam sido ou não referenciadas nos resultados que foram obtidos anteriormente, fazendo assim uma *backward search,* como análise complementar.

Com base nos critérios aqui expostos foram identificados mais de 1000 artigos entre as várias plataformas, dos quais só 207 foram analisados, sendo os mais relevantes referidos ao longo deste trabalho. Inicialmente, para compreensão desta tecnologia já que se tratava de um assunto completamente novo, começou-se pelo conteúdo que especificava o SDN genérico, e das suas questões de segurança. Seguidamente foi analisada a linguagem P4. Finalmente procedeu-se à análise de ataques DDoS em ambientes SDN para melhor compreensão das particularidades destes no quadro geral de segurança de SDN.

Seguidamente procedeu-se à experimentação com a linguagem P4 como base do desenvolvimento da solução DPSynFloodBlock.

2.2 Objetivos do estudo

Segundo o reportado pela NETSCOUT em [72] os ataques de DDoS na primeira metade de 2023, ultrapassaram os sete milhões e meio. Para isso muito tem contribuído a proliferação das *botnets.* Segundo a mesma fonte, os maiores vetores de ataque continuam a ser os ataques de inundação baseados em TCP (SYN, ACK, RST), representando 63% de todos os ataques de DDoS, apesar dos ataques de reflexão/amplificação de DoS, serem os que mais impressionam já que a cada dia novas formas destes aparecem. A título de exemplo, em 2022 foi descoberto um novo vetor chamado TP240 PhoneHome DDoS (CVE-2022-26143) [73] que tem uma taxa de amplificação de 4,293,967,296:1. Para além disto a execução de ataques de DDoS requer um baixo nível de especialização e dificuldade. Daqui percebemos as capacidades dos ataques DDoS na disrupção dos serviços das quais todos nós dependemos e a necessidade de investigação, para encontrar soluções que detetem, previnam e os mitiguem, no âmbito de SDN. Considerando a análise do ambiente SDN, tornou-se evidente que abordar integralmente o problema de segurança no SDN é uma tarefa impraticável para um único indivíduo, especialmente dentro do intervalo de tempo disponível. Diante dessas limitações, optámos por focar o estudo em estratégias de solução no plano de dados, utilizando a linguagem P4 para lidar com ataques de DDoS *Syn Flood*. O objetivo é explorar a viabilidade de desenvolver uma solução não complexa através dessa linguagem, que, além de utilizar informações básicas da arquitetura da rede, dependa exclusivamente dos dados dos fluxos que passam nos dispositivos programáveis. Dessa forma, pretendemos capacitar o plano de dados na deteção e mitigação desses ataques.

2.3 Plano

Para a concretização deste trabalho foram identificadas as tarefas descritas na Tabela II.

Tabela II - Tarefas no âmbito do trabalho.

Tarefa	Descrição
1	Âmbito da tese
2	Análise da arquitetura de SDN
3	Experimentação com a arquitetura SDN
4	Análise da segurança de SDN
5	Análise da linguagem P4
6	Experimentação com P4
7	Seleção de um tópico de segurança para aprofundamento
8	Análise aos ataques de DDoS em SDN
9	Elaboração de uma estratégia de alto nível para a solução
10	Elaboração de relatório intermédio
11	Análise detalhada às soluções existentes em P4 para DDoS em SDN
12	Experimentação com soluções existentes
13	Elaboração de alto nível da solução
14	Especificação dos requerimentos para a solução
15	Implementação da solução
16	Experimentação com a solução
17	Elaboração de relatório final

A Tabela III ilustra a distribuição temporal das tarefas executadas ao longo da elaboração deste trabalho. As tarefas realizadas excederam o prazo inicialmente previsto, estendendo-se por alguns meses além da data de conclusão originalmente estabelecida, devido a compromissos pessoais e profissionais do autor (trabalhador-estudante).

Tabela III - Diagrama de Gantt com o planeamento deste trabalho.

Mês	Setembro	Outubro	Novembro	Dezembro	Janeiro	Fevereiro	Março	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
Tarefa																
Âmbito da tese																
Análise da arquitectura de SDN																
Experimentação com a arquitectura SDN																
Análise da linguagem P4																
Experimentação com P4																
Análise da segurança de SDN																
Selecção de um tópico de segurança para aprofundamento																
Análise aos ataques de DDoS em SDN																
Elaboração de uma estratégia para implementação																
Elaboração de relatório intermédio																
Análise detalhada às soluções existentes em P4 para DDoS em SDN																
Experimentação com soluções existentes																
Elaboração de alto nível da solução																
Especificação dos requerimentos para a solução																
Implementação da solução																
Experimentação com a solução																
Elaboração de relatório final																

Capítulo 3
Contexto

Este capítulo fornece uma visão geral dos conceitos das Redes Definidas por *Software* (SDN - *Software Defined Networking*) para a contextualização dos tópicos abordados nesta dissertação. É apresentada a arquitetura de SDN para a compreensão em detalhe do seu funcionamento. É também apresentado o protocolo P4 e o seu antecessor, para compreensão das comunicações e operações que ocorrem com/e dentro do plano de dados. Adicionalmente são identificados os desafios em termos de segurança que devem ser tidos em conta no desenvolvimento das mesmas, onde serão enumerados os ataques e ameaças à segurança de SDN.

3.1 Arquitetura SDN

O paradigma de Redes Definidas por *Software* permite a programação dinâmica da infraestrutura de rede, onde é feita a separação do plano de controlo do plano de dados e a centralização lógica da gestão, conseguida devido à possibilidade de interação direta dos elementos da rede com o controlador de SDN. Este controlador tem o papel de gerir a lógica relativa ao encaminhamento de dados em todos os dispositivos da rede responsáveis por esta tarefa [7].

A SDN dissocia a forma como as decisões de tráfego são tomadas (ou seja, no plano de controlo) do tráfego que está a ser encaminhado (ou seja, do plano de dados). As decisões de tráfego são tomadas num controlador logicamente centralizado (mas talvez distribuído fisicamente) que funciona como o núcleo de um sistema operativo de rede. Os controladores gerem as configurações de rede e as regras de encaminhamento nos dispositivos de rede através da API da Interface *Southbound* (por exemplo, OpenFlow ou P4Runtime). O plano de aplicação alarga as funcionalidades de gestão do plano de controlo, através da utilização de aplicações, que podem consultar o estado atual da rede ou definir intenções de alto nível que influenciam o estado da rede. No que diz respeito à dissociação de planos, centralização lógica e programabilidade, as propriedades acima mencionadas, distinguem a arquitetura SDN das arquiteturas tradicionais de rede (*networking*).

O desacoplamento da inteligência da rede do *hardware* permite alcançar o objetivo de maior programabilidade no controlo de fluxo de dados, possibilitando avanços em aplicações tais como redes móveis ou internet das coisas (IoT). Mais ainda, permite a rápida implementação e implantação de novos serviços de redes, simplificando o funcionamento e a gestão de todos os protocolos existentes [8] bem como a rápida introdução de novos, sem incorrer em mudanças significativas nos ambientes de rede existentes [9].

A "softwarização" de rede torna possível que estas sejam mais flexíveis, escaláveis e exista um maior controlo sobre os requisitos e comportamentos da rede, obtendo melhor otimização com custo de manutenção de rede mais baixos [10]. Igualmente, para além de uma gestão mais eficiente, esta separação aumenta a agilidade na automação, na provisão,

na extensão, na manutenção e na resolução de problemas nas infraestruturas de rede [11]. Além disso, os dispositivos de comutação passam a ser apenas responsáveis pela recolha e notificação do estado da rede, processamento de pacotes baseados em regras de encaminhamento impostas pelo controlador de SDN. Esta simplificação das funções dos dispositivos de rede torna-os mais fáceis de fabricar e mais baratos [8].

De acordo com a Open Network Foundation, um consórcio sem fins lucrativos dedicado ao desenvolvimento, normalização e comercialização de SDN, a arquitetura SDN considera três planos distintos conforme mostrado na Fig. 1.

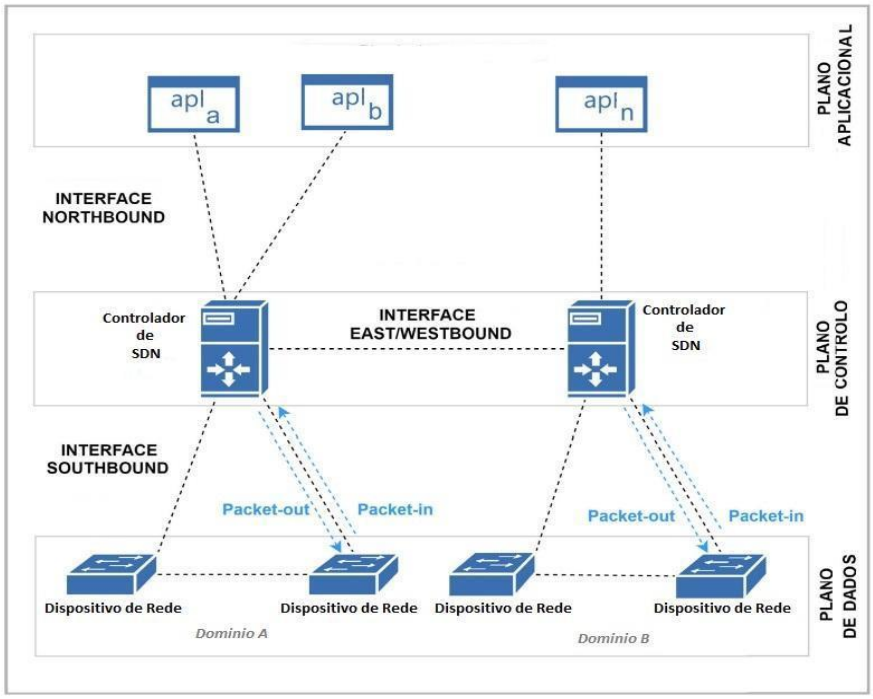


Figura 1 - Arquitetura de SDN. Adaptado de [12].

Plano de Dados

O Plano de Dados, também chamado de plano de infraestrutura, é composto pelo equipamento de rede responsável pelo encaminhamento de pacotes. Encaminhamento que é feito com base na informação de cada pacote (como por exemplo endereços MAC ou de IP, entre outros) e de regras estabelecidas pelo plano de controlo. No caso de um dispositivo não ter regras para um determinado fluxo, o controlador é questionado sobre o comportamento pretendido para esse fluxo específico, que poderá decidir encaminhá-lo ou descartá-lo.

A inteligência da rede é removida deste plano e transposta para um sistema de controlo logicamente centralizado, suportado por interfaces abertos como seja o Openflow ou P4Runtime, que permitem compatibilidade de configuração e de comunicação bem como interoperabilidade entre os dispositivos existentes nos dois planos [12].

Plano de Controlo

No Plano de Controlo está localizado o controlador de SDN responsável pela gestão da rede, que passa por determinar, decidir, configurar e coordenar as políticas de fluxo de

dados necessárias ao funcionamento do DP. Entre elas, decide as tabelas de fluxo e a lógica de transmissão dos pacotes [13]. Este controlador tem uma visão global da rede, suportada através da obtenção de informação dos dispositivos de rede, o que lhe permite ter uma gestão mais abrangente e eficiente dos recursos existentes.

Os dispositivos de rede comunicam com os outros planos através de APIs. Os APIs usados na interface sul ou **Interface *Southbound*** (SBI) permitem que os controladores estabeleçam regras de encaminhamento de baixo nível com esses dispositivos para além de poderem questionar sobre o estado da rede através da recolha de estatísticas [14]. Esta interface especifica os protocolos e as regras para uma comunicação eficiente entre o plano de controlo e o plano de dados [13], como é o caso do SNMP, OpenFlow [15], do NETCONF [16], do OpFlex [17], do OVSDB [18], do ForCes [19] ou ainda do P4Runtime [20] entre outros.

Este plano pode ser composto por um único controlador ou por vários. Neste último caso trata-se de uma centralização lógica de gestão. Os controladores típicos da SDN incluem OpenDaylight (ODL) [21], ONOS [22], NOX, NSX, Nuage, Floodlight (OSS) [23], OpenContrail, Beacon, ONIX, POX, Maestro [24].

A comunicação entre controladores é feita através da interface Este/Oeste, também chamado de **Interface *East/Westbound*,** que permite um sistema de controlo coordenado ou hierarquizado, para diferentes domínios de rede, conseguido através da permissão da comunicação entre controladores ou com outros sistemas SDN, para atualização do estado distribuído [14]. Este estado é composto pelas informações do estado da rede dos vários subdomínios e das escolhas de encaminhamento de cada controlador [25].

A escolha de um controlador deve ser alvo de uma cuidada análise, onde se deve ter em conta os seguintes aspetos [26]:

- Eficácia, que pode ser aferida através do desempenho, escalabilidade, programabilidade, segurança e confiabilidade que oferece.
- Funcionalidades, tais como balanceamento de carga, monitorização de tráfego, aplicação de políticas, introdução de novos serviços, entre outras.
- Aplicação da rede, na medida em que a aplicação especifica os serviços usados na rede. Por exemplo, numa aplicação que é sensível ao atraso, espera-se que o controlador compreenda um serviço que possa fornecer um caminho de ponta a ponta com o mínimo de atraso.

Plano Aplicacional

O Plano Aplicacional alarga as funcionalidades de gestão do plano de controlo através da utilização de aplicações que podem consultar o estado atual da rede ou definir intenções de alto nível que influenciam o estado desta [14].

Estas aplicações suportam o controlador de SDN e alargam as funcionalidades de gestão deste. São essenciais para satisfazer as necessidades do sistema, possibilitando a resposta aos requerimentos do ambiente SDN. Podem ter funções relacionadas com engenharia de tráfego, mobilidade & *wireless*, monitorização e mediação, de segurança e fiabilidade ou ainda relacionadas com redes de *datacenters* [25] [12].

É possível encontrar aplicações tão diversas como seja um *proxy*, um sistema de monitorização de tráfego ou um sistema de deteção de intrusão (IDS) [13].

Tais aplicações comunicam com o controlador de SDN através da interface Norte, ou **Interface *NorthBound*** (NBI), que proporciona a interface para os programadores de aplicações. Esta interface permite que aplicações de rede questionem os controladores para abstrações do estado da rede ou para definir intenções sobre a política de rede [14].

Uma aplicação pode ler ou escrever para uma das *datastores* do controlador através de um serviço correspondente e dos métodos API públicos do serviço. Não existe uma interface padrão para o NBI. A maioria dos controladores suportam várias das diferentes APIs usadas, tais como REST, API, RESTful API, ad-hoc API entre outras [12]. Cada controlador pode estabelecer diferentes limites entre as funcionalidades centrais e as aplicações extensíveis. As aplicações podem ser implementadas de duas formas: como módulos internos dentro do controlador ou como processos externos separados dele (bibliotecas que podem ser manipuladas por exemplo por um programa P4 através de APIs bem definidas, mas cujo comportamento interno é fixo e, portanto, não programável). Um controlador pode ter as duas formas para implementação de aplicações, como é o caso do controlador ONOS, que utiliza a estrutura OSGi em Java para gerir módulos e estados de aplicações internas e API RESTful para aplicações externas [14].

3.2 Benefícios e Desafios da SDN

Benefícios

A arquitetura da SDN tem múltiplos benefícios, entre eles destacam-se [27]:

- Rede Centralizada - Tal como na rede tradicional, as funções de controlo foram fornecidas a cada dispositivo de rede. Mas a SDN centraliza o controlo do fluxo em cada dispositivo e cria uma tabela de fluxo centralizada que é gerida por um controlador SDN central, fazendo com que os dispositivos de rede se comportem apenas como um dispositivo de encaminhamento, onde toda a decisão de encaminhamento está a ser tomada pelo controlador centralizado. Isto simplifica configurações, modificações bem como a resolução de problemas dos dispositivos de rede. Mais ainda, fornece uma plataforma que permite inovar em políticas e protocolos.
- Programabilidade da Rede - Para obter o melhor controlo das deficiências da rede tradicional e proporcionar flexibilidade, a SDN utiliza uma camada de orquestração que controla a rede programaticamente e não afeta o seu desempenho nem fiabilidade. Na rede tradicional as políticas e protocolos são implementados em cada dispositivo, mas em SDN, estas funções de camada de controlo, são separadas e atribuídas a um controlador externo passível de programação o que simplifica a gestão da rede.
- Escalabilidade da rede - O aparecimento de novos serviços, utilizadores e *hosts* requer alterações à rede para comportar os novos requerimentos. A SDN providência a funcionalidade de escalabilidade através da facilidade de introdução de novos dispositivos de rede, sem necessitar do esforço de reconfiguração manual de todos os outros equipamentos. Todas as modificações são programáveis a partir do plano de controlo, que de forma eficiente e fiável as distribuirá aos *targets*.

Nota: Durante este trabalho é referido *switch* ou *target* como uma forma genérica de dispositivo do plano de dados.

- Custo operacional mais baixo - Na rede tradicional, todas as funções são conseguidas utilizando dispositivos de *hardware* físicos distintos, o que requer a instalação de um grande número de equipamentos de *hardware*, aumentando o custo de implantação. Isto, por sua vez, requer mais suporte manual para gerir toda essa infraestrutura física, o que torna o custo ainda maior. A SDN supera este problema reduzindo o custo de implantação, uma vez que todos os dispositivos são controlados programaticamente por um controlador central, além de que um equipamento físico pode ter vários propósitos em vez de um específico, como no caso dos dispositivos das redes tradicionais. Isto permite obter uma rede com menos dispositivos logo menos dispendiosa.

Desafios

Embora a SDN seja uma tecnologia emergente, ainda contém algumas limitações que podem dificultar o seu desempenho. Abaixo enumera-se alguns dos desafios mais importantes deste paradigma [27].

- Fiabilidade - Para aumentar a disponibilidade de rede e evitar erros manuais, o controlador SDN valida inteligentemente as topologias da rede. No sistema de rede tradicional, se os dispositivos de rede falharem, o tráfego da rede é encaminhado através dos dispositivos próximos, mantendo assim a continuação do fluxo. Mas na arquitetura SDN o controlo é centralizado, pelo que se este falhar, todo o sistema pode falhar.
- Escalabilidade - A lógica que diferencia a SDN da rede tradicional é a separação de dados do plano de controlo. O plano de dados SDN e de controlo são dissociados e ambas as funções são independentes, mas estão ligadas através de APIs. Esta separação de planos provoca assim uma limitação de escalabilidade. À medida que o tráfego aumenta, a largura de banda e o número de dispositivos também precisam de ser aumentados, isso terá impacto no processamento do controlador que terá que gerir mais dispositivos, e consequentemente mais fluxos.
- Desempenho sob restrições de latência - O desempenho de SDN é medido utilizando duas métricas, o tempo de configuração do fluxo e o número de fluxos por segundo que o controlador pode suportar. A SDN tem duas formas de controlar o fluxo, pró-ativa e reativa. Se para um fluxo de controlo toda a configuração já existir no *switch* antes da chegada do pacote, quando este chega o *switch* sabe imediatamente para onde o encaminhar. Este fluxo é considerado como fluxo de controlo proactivo. Por outro lado, o modo reativo ocorre quando a configuração não existe à "priori" no *switch*, e este questiona o controlador. Neste caso cabe ao controlador decidir como proceder e para onde o reencaminhar. O tempo para a configuração reativa do fluxo é dado pela soma do tempo de processamento no controlador e do tempo para atualizar o *switch*. Assim, este fluxo limita a escalabilidade da rede e, portanto, introduz um atraso na configuração do fluxo.
- Interface de baixo nível entre o controlador e o dispositivo de rede - A SDN usa aplicações para controlo, gestão e determinar políticas de rede de alto nível. Estas políticas são traduzidas pelo controlador de SDN para uma configuração de baixo nível no dispositivo alvo. O controlador fornece uma interface de programação

que suporta um modelo de baixo nível e orientado para o evento. Esta interface reage à chegada de pacotes e ao estado das atualizações de ligações entre dispositivos (*links*) o que obriga o programador a monitorizar constantemente se as alterações das políticas de comutação afetarão ou não outros eventos futuros.

- Problema de localização do controlador - Todas as formas do plano de controlo são afetadas pelo problema de colocação do controlador. Tanto as métricas de configuração do fluxo, como as de tolerância à falha, ou ainda as de desempenho são também afetadas pelo problema de localização do controlador. Tal significa que a localização (remota ou não) do controlador, bem como o número de controladores necessários, tem de ser feita de acordo com a topologia da rede disponível.
- Segurança - Existe um défice relativo à segurança na SDN. Isto deve-se à ausência de integração com a tecnologia de segurança existente e à sua dificuldade/incapacidade de análise profunda (*deep inspection*) de todos os pacotes. Além disso, melhorar a inteligência do *software* do controlador por si só aumentará a vulnerabilidade do controlador a agentes maliciosos, bem como a sua superfície de ataque. O aumento da utilização de SDN requer um aumento da segurança necessária requerida pelo controlador, para suportar a autenticação e autorização do administrador de rede. Apesar de ser uma tecnologia emergente, ainda carece de políticas padrão que possam compreender topologias, atrasos e perdas. Também não fornece deteção de *loops* e não tem a capacidade de corrigir erros. Além do mais, não contempla colaboração entre os dispositivos de rede, pelo que não permite a comunicação colaborativa entre estes.

3.3 Segurança na SDN

Com a evolução da SDN e a passagem desta para a aplicação prática, as questões de segurança tornaram-se uma das grandes preocupações da sua comunidade. Para passar a ser usada como solução nas redes comuns é necessário garantir as mesmas características de segurança já existentes nas redes tradicionais e se possível resolver os desafios de segurança destas, inovando para criar uma solução diferenciadora. Além do mais, é necessário assegurar a própria SDN e fortalecer a infraestrutura desta.

O paradigma SDN fornece maneiras mais "finas" para detetar e defender ataques à segurança da rede, através da recolha de informações sobre o estado dela e análise do padrão de tráfego. A segurança precisa estar presente em todos os componentes da arquitetura de SDN de forma a proporcionar proteção à disponibilidade, integridade e privacidade da informação [27].

Apesar das vantagens alcançadas através da capacidade de controlar a rede por meio de *software* e da centralização do controlo no DP, estas mesmas características despertam o interesse de utilizadores mal-intencionados e são fonte de problemas para operadores de rede menos preparados. Por um lado, a capacidade de controlar a rede por meio de *software* expõe a rede às vulnerabilidades comuns do *software*, como vulnerabilidades inerentes às linguagens e algoritmos usados. Por outro, a centralização do controlo num único ponto permite que alguém, com intenções maliciosas, consiga afetar direta ou indiretamente o serviço deste e impeça toda a rede de funcionar corretamente, podendo mesmo impedir totalmente o seu funcionamento [28]. Os ataques de bloqueio de serviço (DoS) com foco no controlador representam uma das ameaças mais graves à SDN [1].

Pontos-alvo de ameaças de segurança

Swami et al [29] refere que todos os componentes da arquitetura de SDN têm vulnerabilidades. Mais ainda, todos os planos e interfaces são sensíveis a certos ataques específicos, que podem comprometer não só os componentes da rede que fazem parte do plano aos quais pertencem, como também elementos de outro plano.

A Fig. 2 mostra os vários pontos na arquitetura de SDN suscetíveis a vulnerabilidades e ataques.

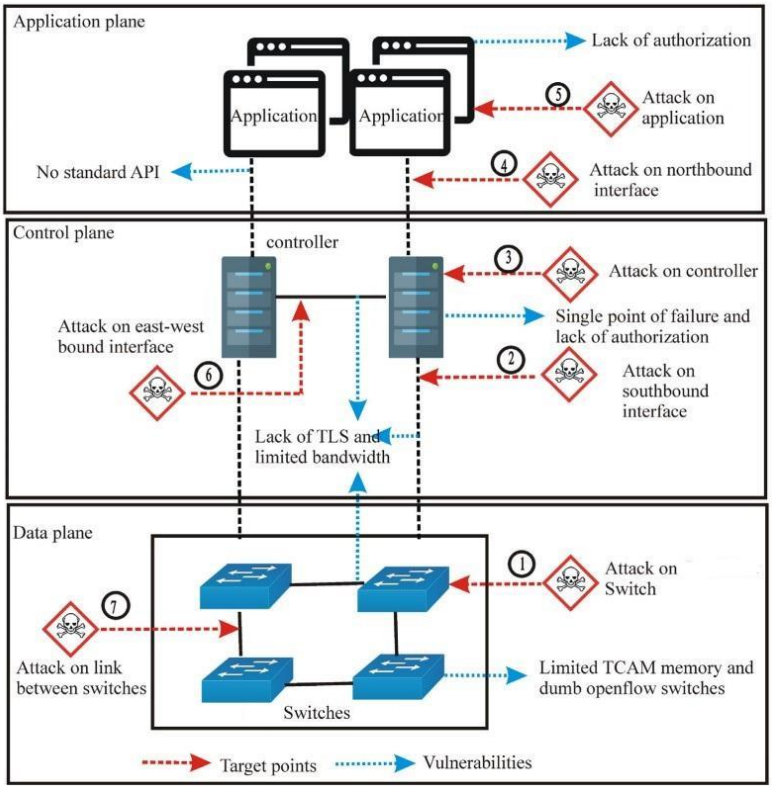


Figura 2 - Pontos vulneráveis na arquitetura SDN [5].

Assim os alvos de ameaças no plano de dados são:

- Dispositivos de rede ou *switches*: Os dispositivos de rede SDN são utilizados para o encaminhamento e processamento de novos pacotes. As vulnerabilidades que afetam os *switches* são diversas. Uma das ameaças inerentes à natureza da arquitetura de SDN relativa aos *switches* são modificações das MAT com impacto malicioso ou negativo. Esta modificação poderá pôr em causa a integridade e a confidencialidade das comunicações. Por exemplo, um agente malicioso pode modificar a regras de encaminhamento nas MAT de forma a desviar o tráfego do seu percurso, permitindo assim efetuar um ataque de *man-in-the-middle* (MITM) [33] para inspecionar as comunicações. Outra ameaça tem que ver com o tamanho das MATs [29] que poderá ter impacto no processamento e memória dos dispositivos de rede, podendo levar ataques de sobrecarga das tabelas de fluxo e

de saturação de buffer causando latência ou negação completa de serviço do *switch* (ataque de DoS) [30].

- Ligações entre dispositivos de rede: Os pacotes são encaminhados de um *switch* para outro. A maioria dos pacotes transferidos não estão codificados e podem conter informações sensíveis. Estes pacotes podem ser intercetados pelos atacantes. Assim existe a ameaça de ataques de MITM [31]. Por outro lado, um agente malicioso poderá enviar pacotes com o cabeçalho alterado fazendo passar-se por outro *switch*, efetuando um ataque de *spoofing.*

Na interface *Southbound* a falta de mecanismos de segurança na arquitetura de SDN para garantir a integridade, autenticidade e a privacidade leva a que seja possível a interceção das comunicações entre o controlador e os *switches* [32]. Desta forma através de um ataque MITM, um atacante fazendo-se passar pelo controlador poderá modificar, apagar ou adicionar regras de fluxo para desviar pacotes, o que poderá ter um efeito devastador no serviço de rede. Similarmente poderá fazer-se passar-se por um dispositivo-alvo, enviando pacotes forjados para o controlador de forma a deturpar a visão global de rede deste ou observar o atraso na comunicação entre o plano de controlo e os aplicativos do plano de dados. Além disso, a limitação da largura de banda deste canal de comunicação torna-o vulnerável a ataques de saturação de largura de banda, como é exemplo um ataque, onde um nó malicioso cria pacotes sem correspondência nas MATs forçando o *switch* a enviá-los para o controlador resultando em congestão no link da interface.

Relativamente ao plano de controlo podemos identificar como alvos de ameaças:

- Controlador: Com a centralização da inteligência no controlador, sendo este responsável por tarefas cruciais ao bom funcionamento da rede, qualquer anomalia do seu serviço tem a capacidade de paralisar a rede. Isto significa que a funcionalidade da rede depende do controlador, tratando-se assim de um ponto de falha crítico (*single point of failure*). Esta característica torna-o o mais apetecível dos alvos de ataque da rede de SDN [29]. Por um lado, a partir de um controlador comprometido (ataque de sequestro) é possível desferir um ataque de DoS ou mesmo efetuar ataques de buraco negro (*blackhole*), um tipo de ataque DoS, onde todo o tráfego é encaminhado para um "buraco negro" fazendo com que todo o tráfego seja perdido. Além disto, ainda é possível inserir regras de fluxo falsas alterando o encaminhamento dos pacotes, o que pode ter como consequência a perda da integridade e confidencialidade de informação sensível. Por outro lado, é possível esgotar os recursos e a capacidade de processamento através do forjamento de pacotes sem correspondência nas tabelas de fluxo obrigando-o a intervir no processo de encaminhamento [33]. No caso de controlo distribuído, ou hierarquizado por muitos controladores, que gerem um subdomínio especifico, a não recolha atempada da informação pode levar a uma gestão da rede deficiente.
- Link entre controladores: Numa arquitetura multi-controladores, a comunicação entre controladores é feita através da interface *east/westbound*, que similarmente às outras interfaces carece de mecanismos para garantir a segurança. Assim, os pacotes entre os controladores podem ser intercetados por um atacante para obter informações essenciais com vista a comprometer os controladores [33]. Além disso, o acesso a este link por parte de intrusos pode provocar falhas em cascata dos controladores devido a inundação de pedidos.

Na interface *Northbound*, que serve de interface para comunicação entre o plano de aplicação e o controlador e é utilizado para recolher estatísticas de rede ou dar instruções ao controlador (para a gestão da rede), a falta de normalização torna esta ligação

vulnerável a múltiplas ameaças, tais como interseção, corrupção/envenenamento ou ataques DoS [33].

Por fim no plano de aplicações, as aplicações desenvolvidas para telemetria, orquestração, monitorização, entre outras, podem ter vulnerabilidades de segurança similares às existentes nas aplicações web comuns, como por exemplo *Cross Site Scripting* (XSS) ou *Cross Site Request Forgery* (CSRF). Estas vulnerabilidades podem permitir a propagação de ataques aos outros planos [34]. Uma aplicação maliciosa pode executar processos no controlador com vista a obter informação sobre toda a rede o que similarmente ao ataque de sequestro do controlador, pode ter impactos devastadores em toda a rede. Mais ainda, as aplicações maliciosas podem enviar pedidos de forma a consumir os recursos do controlador [7]. Além de aplicações maliciosas, este plano é também suscetível a outras ameaças à segurança, tais como aplicações maliciosas, exaustão de recursos e conflitos de políticas.

Vulnerabilidades, Ameaças e ataques

De uma forma mais detalhada Chica et al. em [32] identifica uma lista com os problemas generalizados e ameaças de segurança realçando as falhas de segurança que foram identificadas no contexto SDN:

- Acesso não autorizado: Possibilidade de os agentes maliciosos acederem a componentes da SDN para os quais não têm autorização de acesso nem privilégios de utilização, explorando mecanismos de controlo de acesso fracos ou inexistentes.
- Recolha não autorizada de informações: Obtenção de informação sensível por parte dos atacantes devido a canais de comunicação inseguros, falta de mecanismos de autenticação e controlo de acesso. Por um lado, a falta de encriptação das comunicações permite aos atacantes obter informação das comunicações entre os vários planos e assim perceber a lógica da rede. Por outro, a falha de mecanismos de autenticação permite que os atacantes infiram o comportamento de encaminhamento de rede transmitindo pacotes de sondagem para elementos da rede alvo. Além disto, por falta de controlo de acesso, é possível, usando aplicações vulneráveis, obter acesso à base de dados de políticas e outros dados da rede.
- Modificação não autorizada de informações: Modificação das regras de fluxo por parte de atacantes, colocando em causa o funcionamento da rede, através da exploração de APIs e protocolos vulneráveis, aliados à falta de mecanismos de autenticação, autorização, verificação e responsabilização. Os acessos não autorizados aos recursos da rede possibilitam não só a modificação, mas também a introdução de novas políticas de rede que podem criar conflitos com regras já existentes. Além do mais, é possível alterar a topologia da rede através de injeção de fluxos falsos ou aproveitando vulnerabilidades de protocolo ou personificação de dispositivo de rede. A personificação do controlador, tem consequências desastrosas já que este gere toda a rede [24].
- Destruição de informações de rede: Cenários mais importantes neste caso referem-se à remoção de regras de fluxo dos *switches*, a aplicações maliciosas que descartam pacotes de controlo destinados a outra aplicação ou aplicações e remoção de políticas de rede como consequência de acesso não autorizado concedido a qualquer elemento do plano de aplicação ou à base de dados da rede.

- Interrupção de serviço: A centralização do controlo em SDN, os recursos limitados e a possibilidade de manipulação das regras de encaminhamento, permitem que ataques com objetivo de afetar a disponibilidade do serviço de rede sejam dos mais preocupantes, devido à possibilidade de bloquear completamente o serviço de rede. Entre as muitas fontes de interrupção de serviços possíveis, que resultam na desconexão de dispositivos da rede, destacam-se os ataques de inundação de pacotes de controlo, inundação das tabelas de regras de fluxos, injeção de pacotes malformados e ataques onde o controlador é induzido em erro relativamente à topologia da rede (*topology poisoning attacks*).
- Configurações incorretas: Configurações incorretas em interfaces, protocolos, sistemas e APIs, são traduzidas em novas vulnerabilidades. Assim como políticas de rede e regras de fluxo conflituantes, mecanismos de verificação e autenticação impróprios e também evasão de medidas de segurança já implantadas contribuem para o aumento do risco de segurança.
- Mecanismos de autenticação, confiança e verificação mal configurados: Canais sem encriptação e mecanismos de autenticação fracos ou inadequados potenciam ataques de espionagem, injeção de pacotes, criação de pacotes, intercetação de tráfego, corrupção de informações de rede e sequestro de identidade. Também neste caso esta ameaça poderia ser incluída no ponto anterior já que também se trata de configurações incorretas.

Na minha opinião, a ameaça de destruição de informações de rede pode ser incluída na ameaça de modificação não autorizada de informações, já que o apagamento é uma forma de modificação.

A Fig. 3 mostra as ameaças mais relevantes que podem ser exploradas por agentes mal-intencionados, para comprometer a arquitetura SDN.

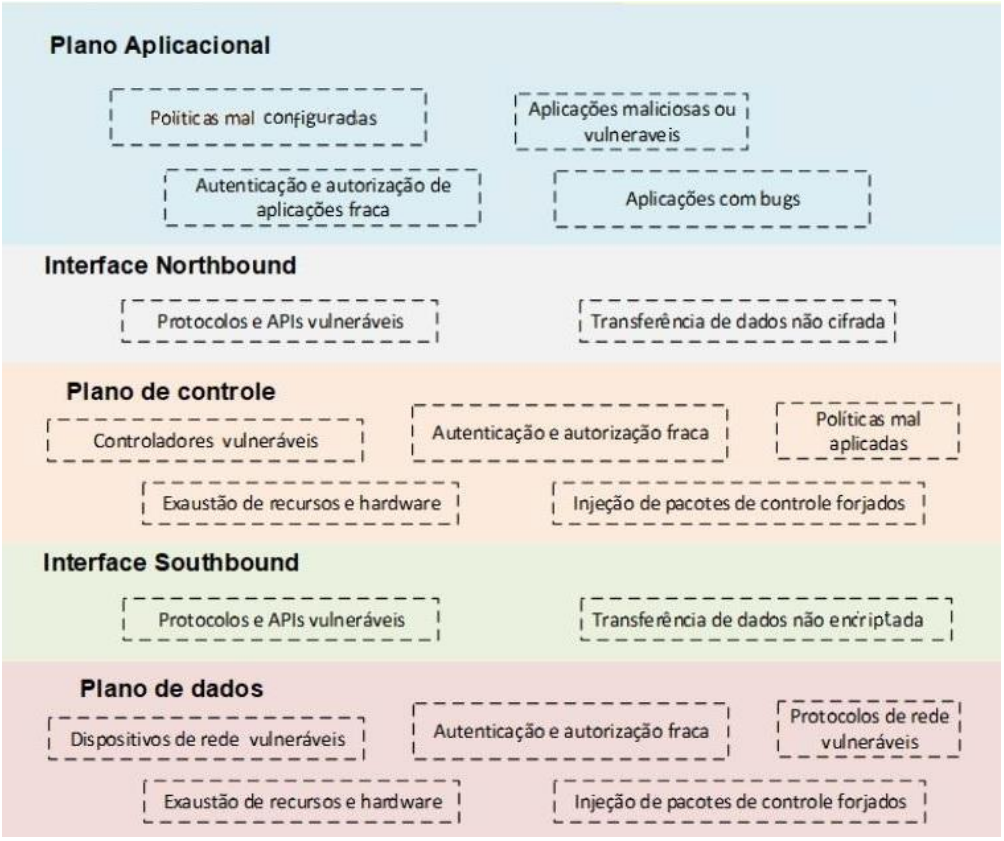


Figura 3 - Ameaças à arquitetura de SDN. Adaptado de [32].

A lista acima confirma os resultados do estudo [35] que diz que existem cinco características que podem ter o maior impacto na segurança das redes SDN, visto aumentarem o número de vulnerabilidades a que as redes estão expostas e que compreendem: o controlador centralizado, as interfaces programáveis abertas, o protocolo de gestão de dispositivos, serviços de rede de terceiros e redes lógicas virtualizadas.

Também no estudo [36] onde foram analisadas e analisadas as severidades de 114 vulnerabilidades genéricas de SDN, concluíram que a SDN tem um número alto de vulnerabilidades médias e críticas, devido às características da sua arquitetura, bem como às fraquezas herdadas da arquitetura de rede tradicional. Sendo que as mais severas estão relacionadas com o controlo de acesso e com as que afetam a disponibilidade. Mais ainda, no estudo calcularam os impactos dos recursos SDN na sua segurança e identificaram os componentes do plano de controlo como os de maior peso, dado que a arquitetura SDN é baseada na separação, na programabilidade e na centralização do plano de controlo. Em contraste, o recurso a elementos do plano de aplicação e de dados têm intensidades mais baixas, porque a SDN não é afetada pelos seus designs. No entanto isto não significa que as vulnerabilidades dos elementos do plano de dados não possam ser exploradas para ataques disruptivos do serviço de SDN, como sejam ataques de DoS, e daí que não sejam ignorados aquando do desenvolvimento de soluções de segurança.

Face ao exposto, os principais ataques à infraestrutura de SDN de acordo com [11] são enumerados na tabela abaixo.

Tabela IV - Principais ataques aos planos de arquitetura de SDN, de acordo com [11]

Camada ou Plano de SDN	Tipo de ataque	Objeto e Impact do ataque
Aplicação	Autoridade não restrita	Término de aplicação.
	Aplicações maliciosas	Execução de comandos de sistema.
	Neutralização de serviço	Manipulação de controladores de pacotes de controle. Execução de uma interrupção de serviço. Captura de informações sensíveis da rede. Execução de ações específicas desviantes
	APIs de *Northbound* vulneráveis	Encerrar uma aplicação vítima. Emitir um comando de sistema. Expor informações trocadas entre o controlador e uma aplicação-alvo.
Controlo	Encapsulamento (ou *tunneling*) de regras de fluxo	Contornar firewall. Instruir regras de fluxo conflitantes.
	Envenenamento do controlador.	Envenenar informações do controlador e visualização da topologia. Propiciar a execução de ataques no plano de dados. Envenenar a topologia da rede com pacotes LLDP falsificados.
	Sequestro de localização do host	Inserir entradas de rede falsas. Envenenar o reservatório de perfis de host do controlador.
	Uso indevido de NOS	explorar configurações incorretas do controlador. Forçar o encerramento do controlador. Redirecionar informações sensíveis. Instalar rootkits. Inserir dados de entrada inválidos deixando o controlador em um estado imprevisível.
	Inundação de Packet_in	Transmitir pacotes malformados em massa. Aumentar as falhas na tabela de switch. Sobrecarregar o controlador respondendo a mensagens packet_in por meio da API Southbound.
	Inundação da tabela de switch do controlador	Receção contínua de pacotes de resposta de características falsificados . Armazenar entradas de switches falsas, na tabela de switches do controlador. Degradar o desempenho do controlador.
	Desconexão forçada de switch	Roubo ilegítimo da identidade do switch ao passar-se por um switch confiável DPID. Envenenar da spanning tree.
	Escuta de canal	Aproveitar canais de controle não criptografados. Realizar sniffing de pacotes. Interceptar o tráfego de controlo, topologia de controle e gestão.
	MITM	Infiltrar comunicações entre o controlador e os dispositivos do plano de dados de destino.
Dados	DoS/DDoS	Exaustão de recursos do switch/controlador. Isolar o switch de destino.
	ARP Poisoning	Roubo da identidade do controlador. Forçar a desconexão pelo switch de destino. Forçar o switch a conectar-se a um controlador falso.
	Manipulação de regras de fluxo	Modificar entradas na tabela de fluxo do switch. Sobrescrever regras de fluxo existentes por meio de uma aplicação de controlador comprometido. Forçar o switch a fazer uma solicitação falsa de regra de fluxo para o controlador.
	Inundação de regras de fluxo	Forçar um switch a solicitar continuamente novas regras de fluxo para inundar a sua tabela de fluxo. Degradar a estabilidade e tempo de resposta do switch
	Injeção de pacotes de controlo	Expor o switch a uma situação de ataque de fuzzing. Injetar pacotes de controle manipulados contendo cabeçalhos malformados em uma tabela de switch.
	Ataques de canal lateral	Inferir informações específicas para realizar outro tipo de ataques (por exemplo, realizar uma inundação de tabela de fluxo em um switch inferindo sobre RTT).

Segurança através da SDN vs. Segurança da SDN

As funcionalidades da SDN, tais como a inteligência centralizada, a programabilidade da rede e a visibilidade geral, remodelaram a forma como as responsabilidades de controlo e gestão básica desta são executadas em redes programáveis. No entanto, tal como detalhado nas secções anteriores, estas características, assim como a própria arquitetura SDN, introduzem novos riscos de segurança e superfícies de ataque, que não estão presentes nas implementações convencionais da rede. Assim, tendo em conta quer os benefícios, quer os riscos, compreendemos dois caminhos distintos a desenvolver para reforçar a segurança da SDN.

1. A necessidade de exploração de recursos e mecanismos de SDN para proteger, reagir e fornecer sistemas de defesa e mitigação contra riscos de segurança bem conhecidos, seja através da introdução de aplicação com novas propostas de segurança, ou do melhoramento das funcionalidades de sistemas e aplicações de segurança já existentes.
2. E por outro, o melhoramento do design da arquitetura SDN, de forma a torná-la mais segura, proporcionando um comportamento proativo contra as novas superfícies de ataque e falhas de segurança contidas em si mesma [37].

Segurança através da SDN

De acordo com [38] para melhorar a segurança através da SDN, podem ser usadas três funcionalidades desta para implementar uma variedade de soluções de segurança no ambiente SDN.

A primeira corresponde ao Controlo Dinâmico do Fluxo, que se diz beneficiar a segurança de duas maneiras diferentes:

- Aplicando a funcionalidade de caixas intermédias de segurança, como uma composição de diferentes conjuntos de regras de controlo de fluxos instruídos por toda a infraestrutura de rede;
- Sob a forma de aplicações de rede, instaladas no topo do controlador ou vinculadas ao controlador através NBI, de forma a não haver necessidade de *hardware* adicional, substituindo-o efetivamente através da aplicação de regras de segurança nos dispositivos de rede genéricos (*commodity network devices*)

Por conseguinte, o controlo dinâmico do fluxo da SDN pode ser alavancado para obter as funcionalidades de *firewalls* locais ou de perímetro, listas de controlo de acesso e esquemas básicos de reorientação de tráfego. Este último mecanismo juntamente com esquemas de segmentação de tráfego, utilizados para distinguir entre tráfego normal e suspeito, introduz a segunda melhoria, que a partir do controlo dinâmico dos fluxos, consegue através da instrução dinâmica das regras de fluxo, forçar os dispositivos de rede a redirecionar certos tipos de tráfego para uma análise mais especializada para sistemas de segurança específicos, ou para o próprio controlador através de uma aplicação de segurança especializada.

A segunda funcionalidade, é a composição da visibilidade de toda a rede com controlo de fluxo centralizado. A visibilidade de toda a rede significa que o controlo da rede pode estar ciente do estado de qualquer elemento desta, instalado em qualquer lugar e em qualquer momento. Por sua vez, o controlo centralizado dos fluxos refere-se à forma como as decisões de encaminhamento de fluxos podem ser tomadas, a partir de uma instância de rede única e logicamente centralizada, aliviando todos os dispositivos da rede de fazer qualquer cálculo ou impor quaisquer algoritmos de encaminhamento aquando da chegada de um pacote de dados. Esta composição de funcionalidades pode beneficiar várias tarefas de segurança, nomeadamente em todos os aspetos de segurança que envolvem a recolha de informações, uma vez que o controlador pode solicitar amostras de fluxo e estatísticas de fluxo usando mensagens específicas implementadas no protocolo da camada de controlo. Estas informações podem ser utilizadas diretamente pelo próprio controlador para instruir qualquer tipo de reação à situação da rede inferida a partir de dados recolhidos ou podem ser disponibilizadas a aplicações de segurança especializadas, como por exemplo um sistema de deteção de intrusão (IDS - *Intrusion Detection System*), Sistema de prevenção de intrusão (IPS - *Intrusion Prevention System*) ou sistema de inspeção profunda ou detalhada de pacotes (DPI - *Deep packet Inspection*). Mais ainda, através da monitorização do tráfego de rede o controlador pode manter registos atualizados sobre o comportamento do tráfego. Nomeadamente, e fluxos de tráfego suspeitos ou malformados que atravessam a infraestrutura, fluxos de tráfego inesperados provenientes de elementos de rede não supervisionados/supervisionados, ou ainda alterações súbitas nas estatísticas de pacotes ou de byte de qualquer dispositivo de rede. A visibilidade, as características recolhidas da rede e a deteção e prevenção de ataques permitem identificar a origem destes últimos. Assim, permitindo ao plano de controlo poder decidir as ações adequadas para mitigar eventuais situações anómalas. Por exemplo, os padrões de tráfego e as estatísticas de fluxo podem alertar para um possível ataque de DoS, proporcionando ao controlador ou certa aplicação de segurança a

possibilidade de invocar as medidas necessárias para detê-lo, quer instalando novas regras de fluxo ou negando completamente qualquer tráfego proveniente da origem de ataque. E por último, a inspeção de conteúdos de fluxo, tais como IDS, IPS e DPI que em redes convencionais podem ser implementados como dispositivos de *hardware* individuais ou como aplicações de segurança baseadas em *software* em execução em servidores multiusos. Também a SDN permite a implementação destes sistemas de segurança de diferentes formas, nomeadamente sob a forma de aplicações do controlador, como serviços de segurança que residem na camada de aplicação, ou como serviços de segurança executados em ambientes *cloud* e até como dispositivos de *hardware* ligados à infraestrutura. Sistemas estes que podem ser alimentados, redirecionando o tráfego diretamente para eles ou através de informações, características e amostras de pacotes extraídos do estado da rede.

A terceira funcionalidade é a programabilidade da rede. Nas redes convencionais, os utilizadores/administradores estão condicionados às caraterísticas específicas dos seus fabricantes, sendo que as reconfigurações e reprogramações apenas podem ser efetuadas através de comandos/instruções proprietários muito limitados. Contrariamente na SDN, devido à programabilidade da rede, a infraestrutura pode ser composta por um conjunto de dispositivos *commodity* em que o cliente pode incorporar diferentes funcionalidades, expressas através de conjuntos específicos de regras de fluxo. Por exemplo, listas de controlo de acesso e filtragem do tráfego podem ser instalados em dispositivos de rede através da imposição de regras de fluxo que descartem ou neguem ações contra características específicas de pacote ou fluxo. Assim, a flexibilidade é potenciada pela programabilidade da rede. Por exemplo, tirando partido da SBI, um operador de rede poderá parametrizar o plano de dados de acordo com os aspetos que descrevem o contexto do funcionamento da rede.

A programabilidade da rede também beneficia da implementação da funcionalidade de uma variedade de serviços e sistemas de segurança como aplicações de rede, que tanto podem ser instaladas como aplicações do controlador ou do plano de aplicação. Permitindo assim a substituição de soluções de segurança limitadas e difíceis de atualizar, por módulos flexíveis de *software* de segurança, que podem ser concebidos para processar quer uma grande diversidade de entradas, quer para oferecer reações rápidas e adequadas, a diferentes situações de ataque. Permite também a construção de soluções de segurança mais robustas e capazes, através do impulsionamento das vantagens e características dos mais recentes avanços da tecnologia de *software*, da inteligência artificial ou de *machine learning*. Pode ainda, ser usada na execução de contra-ataques contra adversários de rede. Um exemplo disto, seria uma aplicação de segurança de rede, que ao detetar um dispositivo malicioso pode ser equipada com algoritmos e *scripts* que lancem um ataque ao *host* malicioso, esgotando os recursos deste, de forma a parar o ataque até que a rede fique salvaguardada deste. Além do mais, a programabilidade da rede introduziu a capacidade de reconfigurar dispositivos de infraestrutura ou de implantar neles funcionalidades diferentes do encaminhamento de pacotes, por exemplo, sistemas básicos de segurança. Isto refere-se à noção de um DP simplificado em SDN, o que significa que a infraestrutura é povoada com dispositivos *commodity*, que podem ser modificados a qualquer momento para cooperar quando necessário com diferentes funções de rede.

Na SDN a dissociação dos planos de controlo do plano de dados significa que um dispositivo de rede depende das decisões instruídas pelo controlador, onde se centra a inteligência da rede. No que diz respeito à segurança, este comportamento pode ser alavancado para implementar serviços e funções de segurança de acordo com determinadas restrições ou políticas de segurança, ou mesmo certas condicionantes de

segurança introduzidos pela conceção do próprio sistema de segurança. Por exemplo, um dispositivo *commodity*, pode ser instruído para negar ou descartar certos pacotes de dados pertencentes a um fluxo de tráfego específico, desta forma implementando listas básicas de controlo de acesso ou esquemas de filtragem de pacotes tirando partido dos recursos de rede já instalados, evitando assim a implementação adicional de *hardware*. As regras de fluxo também podem ser usadas para instruir a infraestrutura na reorientação de fluxos de tráfego suspeitos para sistemas de segurança mais especializados, nomeadamente, aplicações de segurança (quer no controlador, quer em camadas de serviço independentes), mecanismos de segurança e serviços de segurança residentes em ambientes *cloud*. Várias opiniões e estudos recentes, sugerem a incorporação de funções de segurança virtualizadas e cadeias de serviços de segurança virtualizadas na própria infraestrutura, habilitando os dispositivos de rede a desempenhar funções de segurança mais "finas", como filtragem e monitorização de tráfego, inspeção de pacotes, deteção e prevenção de ameaças, isolamento do *host* atacante, aliviando o plano de controlo de parte do processamento de segurança [39]. É neste ponto que o uso de P4, como se verá mais à frente, pode ter um impacto significativo no desenvolvimento de soluções eficazes capazes de dotar o plano de dados de alguma inteligência sem comprometer os pressupostos da arquitetura de SDN.

Segurança da SDN

Por outro lado, para melhorar a segurança da SDN é necessário garantir que quer a sua arquitetura quer o seu funcionamento são suportados por mecanismos que garantam a confiabilidade, integridade e disponibilidade. É, portanto, necessário sanitizar corretamente as ameaças à segurança introduzidas pelas características da arquitetura SDN. Tendo em conta as ameaças e vulnerabilidades apresentadas anteriormente é preciso uma *framework* com comunicações protegidas por encriptação bem como mecanismos de autenticação, autorização, controlo de acesso, responsabilização e verificação. Mais ainda, para garantir uma arquitetura de SDN segura e fiável é preciso que tenha características de tolerância a falhas, recuperação automática, que seja confiável e com capacidades dinâmicas de prestação de serviços. Assim de acordo com [34] os mecanismos de segurança de rede e segurança devem ser incorporados na conceção de uma *framework* de SDN segura são:

- Replicação: Devido a um grande volume de vulnerabilidades de tráfego ou *software* a replicação de aplicações e controladores pode ajudar a lidar com casos de falhas de controlador ou aplicação. A replicação destes elementos pode ajudar a lidar com problemas de *hardware* e falha de *software* (acidentais ou mal-intencionados). Outra vantagem da replicação é a possibilidade de isolamento de aplicações maliciosas, mantendo a consistência do serviço.
- Diversidade: A utilização de apenas um tipo de *software* ou de operação facilita a exploração de um alvo por parte dos atacantes. Em oposição, a diversidade melhora a robustez e a tolerância à intrusão. A utilização de um conjunto diversificado de Sistemas Operativos (OS – *Operation Systems*) torna um sistema menos suscetível a intrusões. A diversidade ajuda a evitar as falhas e vulnerabilidades comuns, uma vez que existem apenas algumas vulnerabilidades comuns entre *software* diversificado ou OS. O uso de controladores diversos pode ajudar a reduzir o movimento lateral de um intruso assim como falhas do sistema em cascata causadas por vulnerabilidades comuns.
- Recuperação automática: Em caso de ataques de segurança que perturbem o serviço da rede, os mecanismos de recuperação de segurança proativos e reativos

podem ajudar a manter a melhor disponibilidade do serviço. Ao substituir um *software*, por exemplo, é necessário realizar a substituição por versões novas e diversas do componente. Por exemplo, se pretendermos mudar o controlador SDN OpenDaylight, podemos considerar uma versão alternativa de *software* controlador, como ONOS, Floodlight, ou ainda o Ryu, que fornecem funcionalidades semelhantes.

- Associação de dinâmica de dispositivos: A associação entre o controlador e os dispositivos deve ser dinâmica na natureza. Por exemplo, se uma instância do controlador falhar, o dispositivo de rede deve ser capaz de associar-se dinamicamente ao controlador de backup de forma segura (implica ter mecanismo de autenticação adequado para detetar um controlador confiável de um malicioso). A função de associação de dispositivos dinâmicos ajuda a lidar com falhas. Além disso possibilita o balanceamento de carga fornecida através de vários controladores reduzindo a latência de serviço.
- Confiança entre controlador e dispositivos de rede: Um mecanismo de estabelecimento de confiança entre o controlador e o *switch* é importante para lidar com casos de fluxos falsos que sejam inseridos por dispositivos de rede maliciosos. O controlador pode, em cenário básico de estabelecimento de confiança, manter uma *whitelist* de dispositivos de rede que são autorizados a enviar mensagens específicas ao controlador. Num cenário mais complexo, pode ser usada Infraestrutura de Chaves Públicas (PKI) para estabelecer confiança entre o plano de controlo e dispositivos de plano de dados. O controlador também pode usar o comportamento dos dispositivos para criar uma *framework* de confiança. Neste caso, dispositivos que mostrem comportamento anómalo podem ser colocados em modo de quarentena pelo controlador.
- Confiança entre plano de controlo e plano de aplicação: É necessário não esquecer que a mudanças no ambiente implica mudança de comportamento dos componentes do *software*. Além disso, o envelhecimento do *software* pode introduzir vulnerabilidades de segurança. Os componentes dos planos de controlo e aplicação devem utilizar mecanismos de gestão de confiança autónoma, baseados na confiança mútua e na confiança delegada (através de *3rd party* - por exemplo, o recurso a Autoridades de Certificados para estabelecer a confiança). O controlador pode utilizar a gestão de confiança autónoma para sistemas de *software* baseados em componentes. Ademais, métricas qualitativas como confidencialidade, integridade e disponibilidade também podem ser alavancadas para estabelecer a fiabilidade de uma aplicação na SDN.
- Domínios de segurança: Os domínios de segurança ajudam a segmentar a rede em diferentes níveis de confiança e a conter as ameaças apenas à secção afetada da arquitetura da SDN. Um isolamento baseado em domínio de segurança pode ser incorporado para fornecer defesa em profundidade para o ambiente de SDN. Por exemplo, a aplicação do servidor web num servidor físico só deve interagir com aplicações de *backend* de base de dados e não com qualquer outra aplicação em execução na mesma rede. Poderá ser utilizada para este fim uma composição da política de segurança baseada em *whitelist*, com um mecanismo adequado de controlo de conflitos de políticas. Similarmente é benéfico usar-se segmentação da criação de políticas, possibilitando o estabelecimento de políticas com granularidade ao nível dos terminais permitindo aos administradores minimizar os privilégios de acesso concedidos a pessoas, aplicações e servidores, e a

limitação do acesso a informações sensíveis ou de missão-crítica com base de apenas o estritamente necessário.

Ataques no plano de dados

Este trabalho foca-se na segurança do plano de dados pelo que seguidamente será feita uma análise dos ataques principais que afetam este plano.

Os dispositivos de rede têm sido um alvo bastante atrativo a ataques. São comprometidos quer através da instalação de *backdoors* por parte de adversários com muitos recursos, quer devido a vulnerabilidades de *software* e *hardware* dos próprios dispositivos ou implementações de protocolos de rede vulneráveis. Estes dispositivos comprometidos podem ser usados para clonar, descartar, tornar lento, desviar, injetar ou forjar tráfego de rede para iniciar ataques com intenção de afetar operadores de rede ou os seus utilizadores. Podendo ser usados para atividades de recolha, exfiltração, manipulação de dados e encobrimento de *tunneling*. Ou ainda para infiltrar VPNs, contornar *firewalls* ou sistemas de deteção e prevenção de intrusão, entre outros.

Considerando as vulnerabilidades do plano de dados, discutidas anteriormente podemos afirmar que de uma maneira geral que este plano é vulnerável à divulgação de informação, negação de serviço e ataques de adulteração (*tampering*). Assim podemos enunciar os principais ataques com origem deste plano [31][34]:

- Ataque de canal-lateral: Correspondente à divulgação de informação, onde por exemplo o intruso pode observar o tempo de processamento do plano de controlo para aprender a configuração da rede. Nestes ataques, um atacante forja pacotes sonda (*probe*) específicos para diferentes camadas de pilha de protocolo de rede, por exemplo, pedidos de ARP para camada MAC e mensagens sonda TTL para a camada IP. Os pedidos podem ser enviados ao controlador juntamente com alguns pedidos de tráfego de base, com uma resposta conhecida (mas que deverá ser reportada ao controlador antes de encaminhado). Observando o tempo de resposta e a diferença de conteúdo entre o tráfego de base e o tráfego *probe*, o intruso pode deduzir o protocolo usado para a comunicação, o tamanho da tabela de fluxo de comutação, registos de comunicação de rede e anfitrião, políticas de monitorização da rede e até a versão do *software* do controlador.
- Ataque de DoS: Correspondente à negação de serviço são considerados a maior ameaça à SDN. Os ataques de saturação do plano de dados para o plano de controlo utilizam falta de correspondência na MAT para inundar os planos de controlo e de dados. Especificamente, um invasor usa vários *hosts* comprometidos (*botnet*) para enviar pacotes massivos a um dispositivo de rede forjando aleatoriamente alguns campos. Não existindo correspondência nas regras de fluxo existentes, os pacotes são colocados no buffer de memória e são enviados ao controlador mensagens *Packet_in* para que este tome uma decisão. Essas mensagens *Packet_in* consumirão uma grande quantidade de largura de banda da interface SBI bem como recursos do controlador (CPU e memória). Por outro lado, mesmo que o controlador decida inserir as entradas de fluxo no dispositivo de rede, pode ser atingido o limite de tabela TCAM desse dispositivo (*overflow*), o que evitará que regras correspondentes a fluxos de tráfego legítimos, como sejam as necessárias para lidar com o ataque, sejam inseridas na MAT do dispositivo devido a esta estar sobrecarregada. Resumindo, os ataques de DoS

podem ser concretizados com recurso à saturação de *buffer* de memória e *overflow* das tabelas de fluxo dos dispositivos de DP, e dos recursos do controlador [25].

- Ataque de corrupção/envenenamento de topologia: Trata-se de um ataque onde são usados pacotes de controlo (LLDP - *Link Layer Discovery Protocol*) forjados para corromper a informação recolhida pelo controlador. Estes pacotes LLDP são normalmente enviados pelo controlador para os dispositivos do plano de dados de forma a recolher informação da topologia de rede. Para o conseguir o controlador poderá usar mensagens *Packet_out* para fazer com que um dispositivo de rede faça *broadcast* de pacotes LLDP que serão posteriormente reportados pelos seus vizinhos ao controlador, permitindo a este último conhecer a topologia de rede. Este ataque do plano de dados é feito em duas fases. No primeiro ataque, o atacante captura os pacotes LLDP e grava a sintaxe LLDP correspondente. No segundo passo, o intruso envia os pacotes LLDP forjados para o controlador ou envia-os para outros dispositivos do plano de dados para ativar a resposta de dispositivo de rede legítimos para controlador de modo a corromper a sua visão de topologia de rede. Isto pode ajudar um intruso a estabelecer uma ligação anteriormente inexistente entre dispositivos de rede, permitindo ao intruso utilizar a topologia modificada a seu favor e lançar ataques de MITM ou ataques de DoS (bloqueando portos legítimos dos dispositivos alvo) na rede.

3.4 DDoS em SDN

O largo espectro de investigação de análise de todos os ataques com foco no plano de dados, torna impossível no tempo disponível tal tarefa, pelo que este trabalho centrar-se-á nas soluções para ataques de DDoS, que são os mais relevantes no contexto deste trabalho.

DoS/DDoS

Um ataque de DDoS é um tipo de ataque de DoS (Fig. 4), que de uma forma generalista é um ataque onde um agente malicioso envia pedidos em largo número para um serviço de forma a sobrecarregar este último, impedindo os utilizadores legítimos de aceder a este ou mesmo a provocar o colapso total do seu funcionamento. Este tipo de ataques tem por objetivo consumir os recursos da vítima e tem consequências extremas para esta. Quando este ataque vem de várias fontes ou origens, designadas como *bots* - que muitas vezes formam uma rede chamada de *botnet* que é controlada por um agente malicioso - é designado como ataque distribuído de DoS (DDoS).

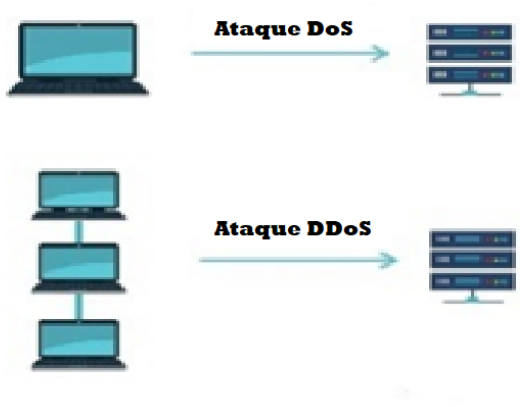


Figura 4 - Ataque DoS vs DDoS.

Os ataques de DDoS podem ser do tipo:

- Ataques volumétricos – onde é enviado um grande volume de tráfego ao sistema da vítima de modo a congestionar a sua largura de banda, como é o caso do ataque de ICMP *flooding*.
- Ataques de exploração de protocolos – onde é explorada alguma característica específica ou falha de implementação de algum protocolo instalado no sistema da vítima. Vulgarmente estes ataques focam-se nas vulnerabilidades da pilha protocolar da camada 3 e 4 do modelo OSI, como é o caso do uso indevido do protocolo TCP, onde são usados pacotes de TCP SYN para efetivar o ataque, chamados de ataques de *SYN flood*.
- Ataques da camada de aplicação – onde são atacadas as aplicações (residentes na camada 7 do modelo OSI) usando inúmeros pedidos que irão exigir uma carga considerável por parte dos servidores o que eventualmente afetará o serviço disponibilizado. E que podem visar os recursos do *host*, da rede, ou da aplicação associados à vítima do ataque.

Além disso, os ataques do DDoS podem ainda ser classificados em termos de aspetos diferentes, tais como ataque baseado em rede (inundação de TCP SYN, inundação UDP), tipo alvo (*host*, *router*, sistema, aplicação), o impacto do ataque (degradante, disruptivo) e tipo de esgotamento (largura de banda, recursos). Assim, existem vários tipos de ataques de DDoS, incluindo inundações de SYN, inundações de ICMP (*ping of death)*, inundações de UDP, inundações de DNS ou ainda DRDoS, DDoS de zero dias e DDoS de baixo débito. Os pacotes UDP/TCP/ICMP gerados por qualquer um dos tipos de ataques acima referidos não só sobrecarregam os recursos de transmissão, computação e armazenamento dos alvos de ataque, como servidores ou controladores SDN, mas também têm impacto na capacidade de transmissão nos dispositivos programáveis do plano de dados que devem lidar com o tráfego excessivo gerado pelos atacantes. Além disso, uma série de ferramentas de ataque DDoS estão acessíveis para download, tornando-se relativamente simples realizar ataques DDoS, tais como Slowloris, Tor's Hammer, Xoic, LOIC, PyLoris, RUDY e DDOSIM entre outras, tornando estes ataques fáceis de desferir, mas difíceis de defender e mitigar [40].

Ataques de DDoS em SDN

Ataques de DDoS são comuns, mas têm um impacto enorme porque afetam o desempenho e o comportamento da rede. De acordo com [40] os ataques DDoS, tendo em conta a arquitetura SDN, podem ser divididos em três categorias:

- Ataques DDoS no plano de aplicação: as aplicações SDN transmitem pacotes específicos para todos ou para a maioria dos dispositivos de rede que suportam o SDN numa tentativa de induzir em erro a aplicação e fazê-la falhar.
- Ataques DDoS no plano de controlo: Os adversários enviam um grande número de novos pacotes para os dispositivos de rede sem correspondência nas tabelas destes, obrigando-os a reenviar para o controlador causando sobrecarga nas capacidades de computação ou largura de banda deste.
- Ataques DDoS no plano de dados: Dispositivos de rede comprometidos transmitem um elevado número de novos pacotes para o dispositivo alvo, de forma a tentar interferir com o armazenamento da tabela de fluxos deste.

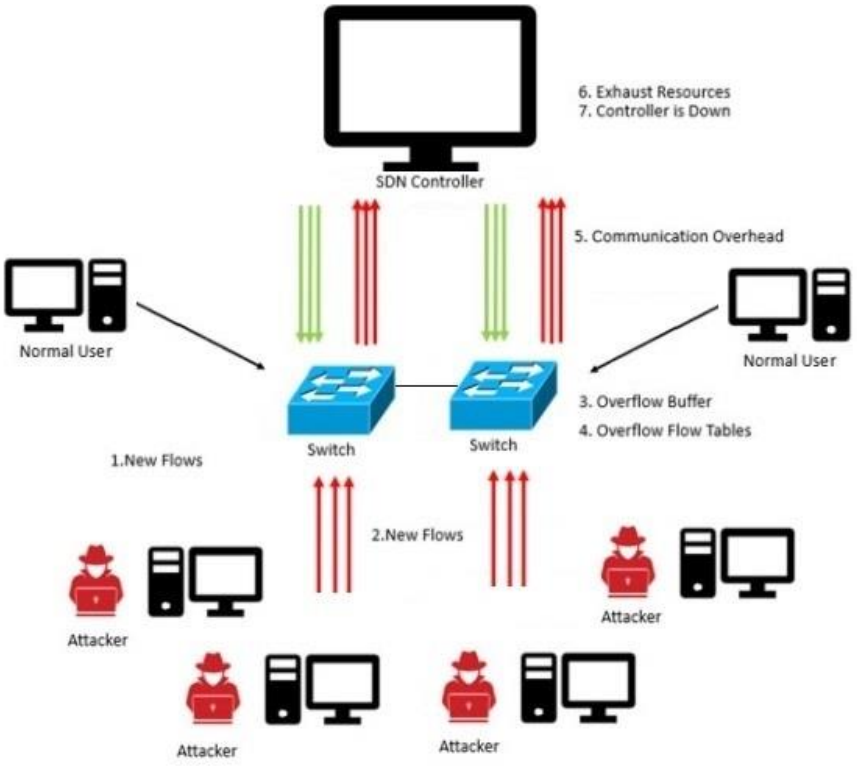


Figura 5. - Ataque de DDoS no plano de dados e no plano de controlo. Adaptado de [41].

Ao esgotar os recursos, os ataques de DDoS impedem ou limitam os serviços da rede impossibilitando as comunicações com o controlador ou o envio de pacotes na rede por parte dos utilizadores legítimos. Numa perspetiva a partir dos dispositivos programáveis - aquela em que este trabalho se foca – os ataques de DDoS relativos no DP e no CP, referidos acima, podem ser conseguidos através da criação de vários novos fluxos que inundariam os dispositivos do plano de dados, a largura de banda das comunicações do plano de controlo e o controlador – como mostrado na Fig. 5 - o que resultaria na falha de rede para utilizadores legítimos. Mais detalhadamente, os atacantes gerariam vários novos fluxos forjando pacotes enviados de várias origens. Estes pacotes com dados

falsificados não teriam correspondência nas regras existentes na tabela de fluxos do dispositivo, resultando numa situação de falha de entrada na tabela. Esta situação levaria à geração de mensagens massivas, de mensagens *Packet_in* ao controlador a partir dos dispositivos vítimas, consumindo largura de banda de comunicação, memória e CPU, tanto no controlo como no plano de dados [42]. Mais ainda, uma vez que os dispositivos enviariam as mensagens de *Packet_in* para o buffer antes de as enviar para o controlador, se vários fluxos novos forem recebidos num curto espaço de tempo, o buffer atinge a sua capacidade máxima. Como consequência seria enviado para o controlador todo o conteúdo dos novos pacotes em vez de enviar apenas os cabeçalhos do novo fluxo, resultando assim num maior consumo da largura de banda do plano de controlo, potencialmente atrasando a instalação de novas regras de fluxo recebidas do controlador. Outra situação que poderia ocorrer neste caso, seria o enchimento da tabela de fluxos dos dispositivos do DP. Assim, os vários fluxos novos resultariam na instalação de novas regras por parte do controlador para a tabela de fluxos do dispositivo, eventualmente resultando na impossibilidade deste último as instalar, descartando os pacotes e enviando mensagens de erro para o controlador. Além disso, o dispositivo não seria capaz de encaminhar os pacotes até que haja memória livre na sua tabela de fluxos, resultando em atrasos e supressão dos pacotes de entrada. Do lado do controlador, uma taxa alta de mensagens P*acket_in* que excedesse a capacidade de processamento do controlador, resultaria em sobrecarregamento deste tornando-o inacessível para o tráfego legítimo. Isto poderia ter como consequência final uma falha a nível da rede, uma vez que o controlador implementa a lógica SDN e gera aplicações e plano de dados [41].

Mecanismo de Defesa a ataques de DDoS em SDN

Tal como anteriormente referido os ataques DDoS em SDNs são problemas graves, e precisam de uma solução real devido aos seus efeitos no desempenho de SDN. Tendo presente o discutido anteriormente relativamente à segurança em SDN e de acordo com [42], as soluções que abordam ataques DoS/DDoS podem ser classificadas com o foco nos componentes dos SDNs e nos seus elementos funcionais, chamadas de soluções intrínsecas; nos fluxos de rede e nas suas características, chamadas soluções extrínsecas ou externas; e adicionalmente baseadas em dispositivos de rede inteligentes ou em canais de comunicação seguros.

Estas duas primeiras categorias ainda podem ser classificadas em subcategorias como apresentado abaixo. As subcategorias das soluções intrínsecas são soluções de entrada de tabela, soluções baseadas em agendamento e soluções baseadas em arquitetura, enquanto as soluções extrínsecas são sub-classificadas em soluções baseadas em estatísticas e soluções de *machine learning* ou aprendizagem automática [43].

- **Soluções intrínsecas**

As soluções intrínsecas são as abordagens que se focam nos componentes de SDN e na funcionalidade dos seus elementos. São classificadas em abordagens baseadas em tabelas, agendamento e arquitetura. As suas sub-categorias são:

- Soluções de entrada de tabela

  As abordagens baseadas na tabela propõem soluções relacionadas com o tamanho limitado das tabelas para os dispositivos do DP. Estas tabelas podem ser tabelas de

buffering (memória) de fluxos desconhecidos ou tabelas de encaminhamento que contenham as regras de fluxo. Cada fluxo desconhecido precisa de uma nova entrada na memória do dispositivo — introduzindo um estrangulamento durante os ataques DDoS, onde existem pacotes com diferentes endereços IP que geram fluxos maciços sem correspondência nas tabelas. Para resolver este problema, várias características foram tidas em conta, nomeadamente:

- Análise/recolha de estatísticas e monitorização do tráfego: esta função indica que a solução proposta utiliza a análise do tráfego de entrada para detetar anomalias. Isto é conseguido através da monitorização do tráfego e da recolha de estatísticas específicas relacionadas com as suas propriedades (por exemplo, número de pacotes por fluxo, número de pedidos dentro de um intervalo de tempo). Com base nos dados recolhidos, as estatísticas são então comparadas com um conjunto de limiares predefinidos para detetar anomalias.
- Ferramentas: tal funcionalidade indica que a solução proposta utiliza algumas ferramentas como Sistemas de Deteção de Intrusões (IDS) ou ferramentas de análise para deteção de ataques.
- Gestão de falhas em tabelas: uma das abordagens para lançar o ataque DDoS é saturar a memória do dispositivo de rede com pacotes, desencadeando processos sem correspondência. Por conseguinte, algumas soluções baseiam-se na gestão de situações de falta de correspondência na tabela dos novos fluxos de entrada, de modo que o controlador SDN esteja protegido antes que o ataque se propague pela rede.
- Gestão da tabela de fluxo: esta função indica que a abordagem proposta impede a saturação das tabelas de fluxos localizadas nos dispositivos de rede. Estas inundações ocorrem quando o controlador instala novas regras de fluxo no dispositivo para encaminhar o grande número de novos pacotes de entrada devido ao ataque. As abordagens utilizando esta técnica, dependem da gestão das regras de fluxo, eliminando ou priorizando-as de modo que as tabelas dos dispositivos estejam protegidas contra saturação.
- Gestão da largura de banda: O objetivo desta solução é limitar a taxa de pedidos de entrada ao controlador ou aos dispositivos para os proteger contra ataques DoS/DDoS.
- *Machine Learning*: significa que a abordagem envolve mecanismos de aprendizagem automática na deteção de ataques.
- Algoritmo do controlador: indica que a abordagem implementa um algoritmo que está a funcionar dentro do controlador. O algoritmo tem acesso a alguns dados dentro do controlador e utiliza estes dados para tomar decisões relevantes para a prevenção, deteção ou mitigação de ataques.
- Controladores múltiplos: indica que a abordagem se baseia na utilização de múltiplos controladores que atuam e operam entre si para abordar o ataque.
- Bloqueio de origem de ataque após deteção: indica que a solução rastreia a origem de ataque, bloqueando-a uma vez detetada.
- Regras de fluxo após deteção: indica que a solução instala regras de fluxo nos dispositivos de rede para controlar o ataque uma vez detetado.

- Tipo de abordagem: esta característica mostra se a solução proposta é para prevenção de ataques, deteção, mitigação ou para continuar a trabalhar em situação de ataque.
- Implementação: tal característica mostra a metodologia de implementação da solução proposta. Em particular, se for baseado em emulação Mininet, implementação de *hardware* ou métodos de implementação de *software* virtuais ou outros.

➢ Soluções baseadas em agendamento

São soluções intrínsecas destinadas a proteger o controlador de ser sobrecarregado e a gerir as redes e recursos da SDN. Para resolver este problema, várias características mencionadas nas *soluções de entrada de tabela*, são também usadas neste caso conforme se pode verificar na Tabela V. Adicionalmente às já faladas são usadas mais duas características:

- Agendamento: tal funcionalidade indica que a solução utiliza o agendamento de pedidos, pacotes, fluxos do novo tráfego de entrada ou agendamento de pedidos de entrada direcionados do dispositivo de rede para o controlador.
- Partilha de Recursos: tal funcionalidade indica que a abordagem proposta gere os recursos partilhados entre os *hosts* ou entre os dispositivos de rede. Alguns exemplos destes recursos são a largura de banda do canal de comunicação ou os ciclos de processamento do controlador.

➢ Soluções baseadas em arquitetura

A última classe de soluções intrínsecas são soluções baseadas em arquitetura. Muitas das soluções propostas nesta subcategoria são baseadas no desacoplamento da monitorização e controlo das propriedades do controlador e o alargamento da arquitetura SDN. As abordagens usadas são mostradas na Tabela V.

❖ **Soluções extrínsecas**

As soluções extrínsecas focam-se nos fluxos de rede e nas suas funcionalidades. São classificados como estatísticas ou soluções baseadas em aprendizagem automática [43]. As seguintes sub-categorias fornecem diferentes exemplos destas soluções.

➢ Soluções baseadas em estatísticas

Em primeiro lugar, as soluções baseadas em estatísticas visam proteger o controlador através da análise e recolha de estatísticas relacionadas com os fluxos. As abordagens usadas são mostradas na Tabela V. Adicionalmente às já descritas são usadas mais três características:

- Verificação do Cliente: tal característica indica que a solução verifica a identidade do cliente antes de permitir que estabeleça uma ligação com a SDN. Esta verificação poderia ser conseguida através de várias técnicas, tais como a Prova de Trabalho, a verificação das ligações TCP e a aplicação de um processo completo de aperto de mão.
- *Firewall*: tal característica indica que a solução envolve uma firewall que é usada para deteção ou mitigação de ataques DDoS

- Redirecionamento de tráfego: neste caso o tráfego é redirecionado para um servidor específico, uma base de dados ou sistema de deteção de intrusão.

➢ Soluções baseadas em Machine Learning

As soluções baseadas em aprendizagem automática fornecem outras abordagens para a identificação dos ataques de DDoS em SDNs. Neste tipo de abordagem, o mecanismo de defesa é um algoritmo de aprendizagem automática que está a treinar nos fluxos livres de ataque para ser capaz de detetar anomalias no tráfego. Estas soluções baseiam-se nas abordagens mencionadas na Tabela V para este tipo de soluções.

Tabela V - Soluções a ataques DDoS em SDN vs Abordagens. Baseado em [41]

Abordagens \ Soluções		Soluções Intrínsecas			Soluções Extrínsecas	
		Soluções de entrada de tabela	Soluções baseadas em agendamento	Soluções baseadas em arquitetura	Soluções baseadas em estatísticas	Soluções baseadas em Machine Learning
Análise/recolha de estatísticas e monitorização do tráfego		✓	✓	✓	✓	✓
Ferramentas		✓	✓	✓	✓	
Gestão de falhas em tabelas		✓	✓	✓		
Gestão da tabela de fluxo		✓		✓	✓	
Gestão da largura de banda		✓	✓	✓		✓
Machine learning		✓			✓	✓
Algoritmo do controlador		✓	✓	✓	✓	✓
Controladores múltiplos		✓		✓		✓
Bloqueio de origem de ataque após deteção		✓		✓		
Regras de fluxo após deteção		✓		✓		
Tipos de abordagem	Prevenção	✓	✓	✓		
	Deteção	✓		✓	✓	✓
	Mitigação	✓	✓	✓	✓	✓
	Sob-ataque		✓	✓	✓	✓
Regras de fluxo após deteção		✓	✓	✓	✓	
Implementação	Virtual /Outro SW		✓	✓	✓	
	Mininet	✓	✓	✓	✓	
	HW	✓	✓	✓	✓	
Agendamento			✓	✓		
Partilha de recursos			✓			
Verificação do cliente					✓	
Firewall					✓	
Redirecionamento de tráfego					✓	

❖ **Dispositivos de rede inteligentes**

Algumas soluções baseiam-se em tornar os dispositivos mais inteligentes para serem capazes de detetar ou mitigar melhor os ataques. Outras soluções envolvem verificação da ligação ao cliente para detetar fontes com endereços IP falsificados.

❖ **Canais de comunicação seguros**

As soluções que envolvem assegurar o canal de comunicação entre os planos de dados e de controlo para abordar os ataques DDoS empregam técnicas de segurança dos protocolos, encriptação e segurança da camada de transporte (TLS) para proteger o canal de comunicação. A utilização de tais mecanismos proporciona uma proteção mais fiável da rede. No entanto, a integração destes protocolos no ambiente SDN é um desafio devido à necessária integração destes mecanismos com os requisitos de SDN. Outras soluções usam o SBI para reduzir a sobrecarga do controlador para protegê-lo de falha devido aos ataques. Isto pode ser conseguido limitando a taxa de envio de mensagens de pacote ao controlador. A utilização do canal de comunicação para abordar os ataques DDoS ajuda a diminuir o impacto do ataque reduzindo a sua intensidade. Consequentemente, há menos pedidos ao controlador que podem esgotar os recursos de rede.

É preciso ter em consideração que ataques de DDoS sofisticados para além de usarem menos recursos do que os despendidos pelo alvo, simulam o comportamento normal do tráfego, variando a taxa deste, de forma a evadir a deteção aumentando o sucesso destes, mas dificultando as técnicas de defesa contra eles. Como qualquer ação para ultrapassar estes ataques passa pela deteção dos mesmos, é necessário identificar as dificuldades na deteção destes ataques, que segundo mencionado em [1] são:

- Recolha de dados estatísticos: A maioria das técnicas de deteção de ataques DDoS precisam de recolher dados dos dispositivos de rede para construir as suas abordagens, tais como a extração das características necessárias do cabeçalho do pacote para detetar comportamentos anómalos. Com o aumento do volume de ataque de DDoS, a recolha de dados estatísticos dos fluxos de tráfego torna-se mais difícil e desafiante, especialmente quando envolve ataques DDoS de baixo débito. Apesar de existirem técnicas que distribuem as tarefas de recolha de dados através de múltiplos dispositivos de rede para equilibrar as cargas de recolha de dados, isto dificulta a recolha destes.
- Seleção de algoritmos: Devido à diversificação do comportamento de ataque do DDoS a deteção de tráfego anormal no ambiente SDN é dificultada. Por esta razão, para detetar comportamentos de ataque de DDoS muitos algoritmos recorreram à rede neuronal artificial, classificação bayesiana, lógica difusa, etc., apesar de até ao momento não haver um único algoritmo que seja capaz de lidar com todas as variações dos comportamentos de ataque do DDoS.
- Resposta imediata: Para o controlador manter a disponibilidade da rede é extremamente importante que seja dada uma resposta imediata ao ataque do DDoS, pois a quantidade massiva de tráfego a chegar ao controlador poderia esgotar todos os seus recursos inibindo a sua capacidade de responder a pedidos legítimos.

Muitas das técnicas existentes para superar as dificuldades de detetar ataques de DDoS centram-se no controlador. No entanto, os problemas com estas abordagens são muitos, entre os quais o pesado encargo do controlador quer para processar um número esmagador de pacotes de entradas num curto espaço de tempo quer pelo processamento necessário de resposta a pacotes inválidos; elevado consumo de recursos de rede; e dificuldade em detetar ataques DDoS de baixo fluxo de tráfego, que simulam o tráfego legítimo e normalmente resultam numa elevada taxa de falsos positivos [44]. Assim, soluções para abordar ataques de DDoS devem passar não só pela combinação das diferentes abordagens anteriormente expostas para obtenção de uma solução mais efetiva, mas

também pelo desenvolvimento de soluções que aliviem o controlador. Isso passa por implementar soluções no plano de dados e na interface deste com o controlador.

3.5 Programabilidade do Plano de dados

Openflow

Inicialmente proposto por McKeown et al. em [45], o Openflow foi anteriormente considerado o protocolo padrão para comunicação entre o plano de controlo e o plano de dados permitindo a abstração necessária para a gestão centralizada inerente ao SDN, focando-se no controlo do encaminhamento de pacotes. Este protocolo tira partido do facto dos comutadores e *routers* modernos terem tabelas de fluxo para funções de roteamento, máscaras de sub-rede, proteção de firewall e análise estatística do fluxo de dados [8]. Além disso, este protocolo, de código aberto, permite que mesmo dispositivos com design e interfaces proprietários [46] sejam administrados remotamente, o que aliado ao seu baixo custo permitiu uma aceitação quer do meio académico, quer do mercado.

O controlo do comportamento de encaminhamento é conseguido através de tabelas de fluxo (Fig. 6).

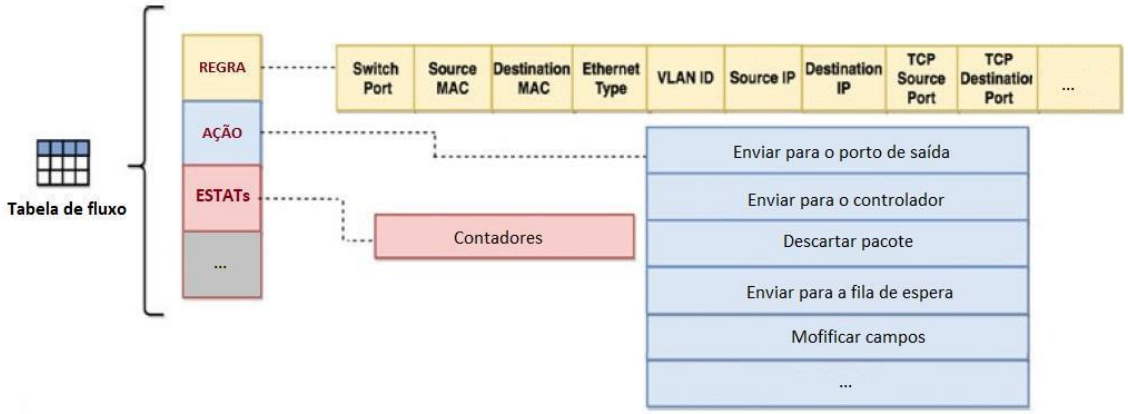


Figura 6- Tabela de fluxo de protocolo OpenFlow.

Na sua versão simplificada, cada tabela de fluxo consiste em 3 componentes: regra (*rule* ou *match*), ação (*action*) e estatísticas (*stats*). A configuração do encaminhamento de pacotes é feita através da correspondência (*matching*) dos fluxos, que são caracterizados por determinados atributos de pacotes de entrada, às quais são associadas ações (*action*) específicas de encaminhamento [47]. Existem ainda as estatísticas correspondentes a esses fluxos que suportam a tomada de decisão do controlador. No caso de um fluxo não ter correspondência na tabela o dispositivo de rede envia uma mensagem de *Packet_in* para o controlador que decidirá a ação a aplicar relativa a esse pacote e que será respondida através da mensagem P*acket_out* ou *Flow_mod*. O controlador solicitará informações sobre o estado do dispositivo através de mensagens *Read-State* e fará alterações nas tabelas de fluxo dos dispositivos de rede através de mensagens *Modify-State* [48].

Ao longo da evolução do protocolo tanto os campos das tabelas de fluxo, bem como os atributos que caracterizam o *match* ou mesmo o número possível dessas tabelas, foram

adicionados para fazer face às limitações de programabilidade. No entanto estas limitações subsistiram, por um lado devido às regras de encaminhamento serem restringidas a um número fixo de atributos/protocolos, e por outro devido ao facto do plano de dados ser limitado às funções fixas do design dos chips. Para fazer face às restrições do protocolo, cedo se percebeu que era necessário dotar os dispositivos de alguma "inteligência" transformando-os em dispositivos também capazes de efetuar operações internas baseadas no seu estado (*stateful*) e assim melhorar a desempenho da rede.

Para ultrapassar as limitações do Openflow e fornecer abstrações de programação mais expressivas, é necessário o desenvolvimento de um DP *stateful* que satisfaça os seguintes princípios [35]:

- Manter a informação do estado dos fluxos dentro do dispositivo de rede e permitir a transição do estado do pacote formalizada programaticamente;
- Oferecer a capacidade ao dispositivo (*target*) de tomar decisões de encaminhamento, com base nas informações de estado dos fluxos locais, sem a necessidade de entrar em contacto com o controlador.

P4

O P4 (*Programming Protocol-independent Packet Processors*), proposto por P. Bosshar et al. [49] e posteriormente desenvolvido pelo P4 *Language Consortium* (p4.org), é uma linguagem de programação de alto nível, específica de domínio, que gera a abstração necessária para um encaminhamento de pacotes flexível, reprogramável e independente do *hardware* subjacente. O comportamento do plano de dados é definido de forma similar ao OpenFlow, o que, de maneira simplista, significa definir quais os campos a combinar e que ações devem ser realizadas. Não obstante, difere deste porque permite alcançar os princípios de um DP *stateful* descritos acima, reduz a comunicação DP-controlador e consequentemente a latência introduzida por esta. Para isso, contribui a utilização de registos de memória (*registers*) que podem ser acedidos durante o processamento de um pacote, permitindo assim a definição de comportamentos de encaminhamento *stateful* [50].

Até ao momento apenas dois padrões distintos de P4 foram padronizados: P414 e a P416. O padrão atual é a versão 1.2.3 da P416 [52] de 2022 que foi introduzido para colmatar a falta de meios para descrever vários *targets* e arquiteturas. Devido à sua semântica simples e precisa, o P416 permite a implementação dos protocolos de rede já existentes, e acelera a implementação de protocolos inovadores e de aplicações com design único, que podem ser escritos e implementados não somente pelos fabricantes de dispositivos de rede, mas também pelos operadores destas. Em oposição ao OpenFlow, o seu uso não é restrito a dispositivos com função de rede fixos, suportando em contraste, uma grande variedade de plataformas de dispositivo (ASICs, FPGAs, NICs ou *software*). É adequado para plataformas neutras - ou de uso não especificado - de equipamentos de rede, permitindo um desacoplamento da evolução do *software* relativa ao *hardware* [50]. A Fig. 7 mostra a evolução da arquitetura de rede tradicional até ao modelo SDN com DP programáveis com recurso a P4.

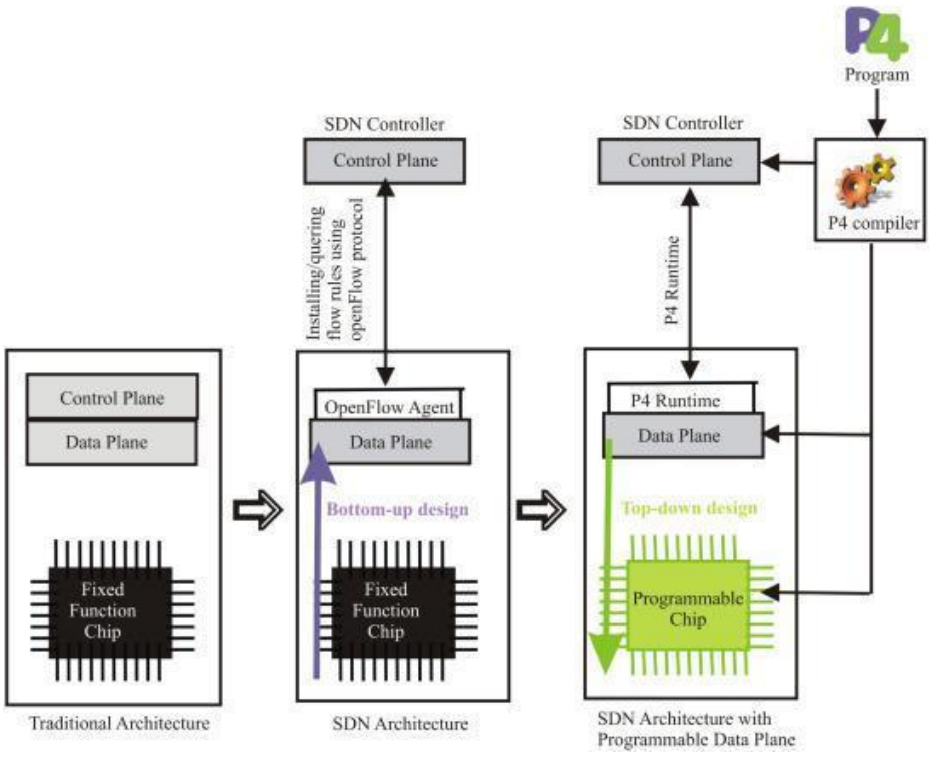


Figura 7 - Evolução da arquitetura de rede tradicional para o modela SDN com planos de dados programáveis [33].

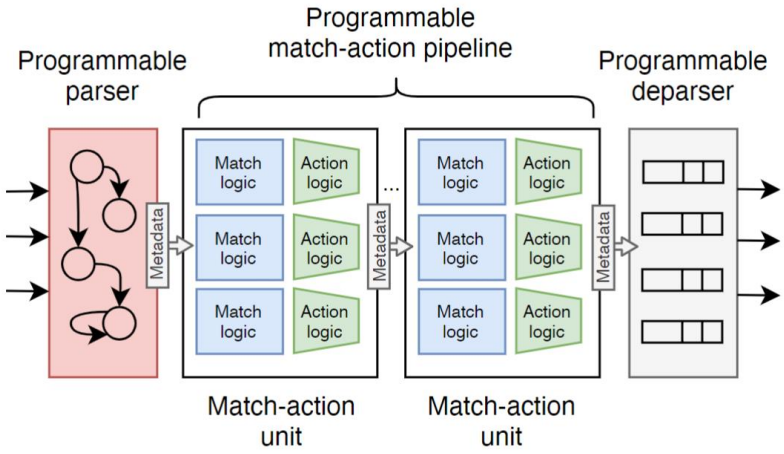


Figura 8 - PISA: Protocol-independent Switch Architecture [53].

O P4 baseia-se no modelo PISA (*Protocol-independent Switch Architecture*) representado na Fig. 8, e compõe-se por 3 blocos programáveis [33]:

- *Parser* ou máquina de estados, é responsável pela identificação da pilha de protocolos e campos permitidos definidos pelo programa. É representado por uma máquina de estado finita, onde os estados são responsáveis pela extração da estrutura do cabeçalho individual em variáveis executáveis. As transições podem ser feitas de maneira condicional com base no valor de um campo de cabeçalho analisado.

- *Pipeline Match-action*, ou *pipeline* de processamento, é responsável pelo processamento condicional do pacote. Contém várias tabelas de ação de correspondência ou *match-action* (MAT), que com base em diferentes campos de cabeçalho do pacote e/ou metadados, decidem que ações devem ser executadas quando ocorre uma correspondência. Também define uma função de controlo, que descreve o processamento de pacotes dependentes de dados dentro de um pipeline e que especifica a ordem pela qual as tabelas devem ser executadas. Cada MAT inclui a lógica de correspondência acoplada à memória estática de acesso aleatório (SRAM) ou à memória ternária endereçável ao conteúdo (TCAM), para armazenar chaves de pesquisa e os dados de ação correspondentes. A lógica de ação (por exemplo operação aritmética ou modificação de cabeçalho) é implementada por unidades lógicas aritméticas (ALUs) e é gerida pelo plano de controlo através da manipulação das MATs. Adicionalmente pode ser implementada utilizando objetos *stateful*, como sejam contadores ou registos (armazenados na SRAM), ou ser fornecida pelo plano de controlo em tempo real [53].
- *Deparser*, ou montagem de pacote, reconstrói o pacote, serializando-lhe os cabeçalhos modificados para que seja enviado para o próximo dispositivo.

Um pacote, processado por um pipeline PISA, consiste na carga útil (*payload*) do pacote, cabeçalho de protocolos de rede e metadados deste. De salientar que a *payload* do pacote não é alvo de processamento pelo PISA. Podemos dividir os metadados em intrínsecos ou definidos pelo utilizador.

Metadados intrínsecos são metadados que se relacionam com os componentes de função fixa. Estes componentes de função fixa comunicam com os programáveis gerando e/ou consumindo estes metadados. Alguns exemplos destes componentes de função fixa são blocos de entrada e saída (*ingress/egress*) que recebem ou enviam pacotes, ou blocos de processo de replicação de pacotes que implementam *multicasting* ou clonagem/espelhamento de pacotes, ou ainda gestores de tráfego, responsáveis pelo *buffering*, *queueing* e programação de pacotes. Um exemplo desta relação é a geração, por parte de um bloco de porto de entrada, de metadados *ingress* que representam o número do porto de entrada e que pode posteriormente ser usado dentro das unidades de *match-action.* Outro será na produção de um pacote, onde as unidades de *match-action* geram metadados intrínsecos, que representam os números de portas de saída e que serão consumidos pelo gestor de tráfego e bloco de porto de saída.

Metadados definidos pelo utilizador são frequentemente referidos como simples metadados que servem de armazenamento temporário, semelhante a variáveis locais noutras linguagens de programação. Através deles, os programadores podem adicionar informações aos pacotes, que poderão ser utilizadas ao longo da cadeia de processamento.

Todos os metadados são transitórios, o que significa que são descartados quando o pacote correspondente sai do *pipeline* de processamento.

Apesar de ter semelhanças com o OpenFlow, o P4 difere deste na medida em que não limita os campos de cabeçalho, nem condiciona a seleção de unidades *match-action* a correr em série. Neste caso a seleção *match-action* pode ser feita em série, em paralelo, ou ambas [35].

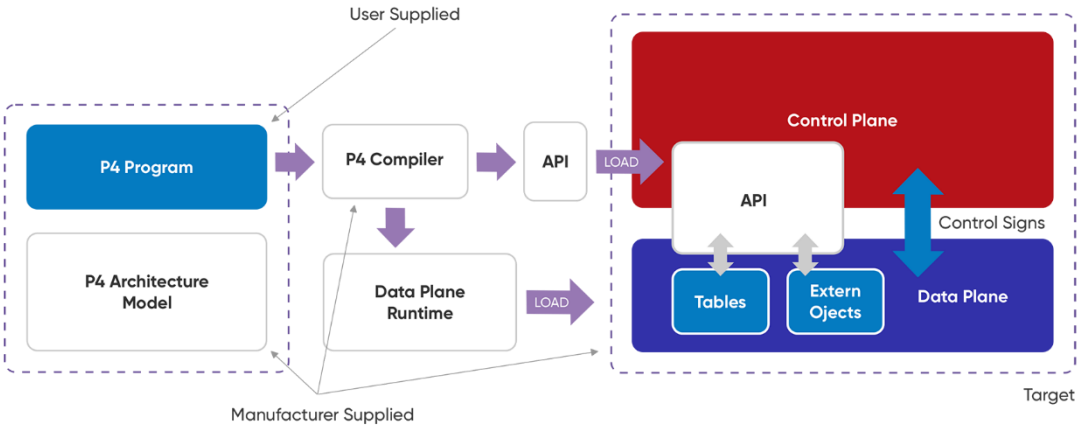


Figura 9 - Processo para programar um dispositivo de rede usando P4 [3].

O algoritmo completo do plano de dados do P4 dispõe de *pipelines* de processamento de pacotes, constituídos por componentes P4 programáveis e de função fixa. A estrutura exata destes *pipelines* é dependente dos *targets* e é determinada pelos seus fabricantes. Esta é descrita por um modelo de arquitetura P4 específico ao dispositivo. Igualmente também os compiladores P4, são dependentes dos *targets*. Estes compiladores, fornecidos pelos fabricantes, têm como função traduzir os programas P4 em código específico para o *target*, que será posteriormente carregado e executado nele. P4c é um compilador P4 de referência [54], que suporta ambas as versões de linguagem P4, P414 [55] e P416 [52]. Por outro lado, os programas P4, são fornecidos pelos programadores [53]. Estes programas definem por um lado os algoritmos que serão executados pelos componentes programáveis P4 e por outro a interação com os componentes implementados na lógica da função fixa. De salientar, que tais programas podem ser implementados em todos os *targets* que instanciem a mesma arquitetura P4. Os *targets* podem ser baseados em *hardware* (por exemplo Barefoot Tofino, FPGA) ou baseados em *software* (por exemplo BMv2, BPF/XDP, PISCES ou P4rt-OVS) [50].

Na Fig. 9 observa-se o processo de programação P4 de um dispositivo de rede (*target*) disponível como *software* ou *hardware* especializado, onde se evidencia que através do suporte do compilador, o P4 gera não só as funcionalidades necessárias ao DP mas também a API necessária para a comunicação do plano de dados com o de controlo, permitindo-lhe não se preocupar com detalhes de implementação específicas dos dispositivos.

As arquiteturas P416 mais comuns são:

- ***SimpleSume Architecture***, como mostrado na Fig. 10 é uma arquitetura P4 simplificada que é implementada por *targets* P4 baseados na FPGA (*Field Programmable Gate Array*).

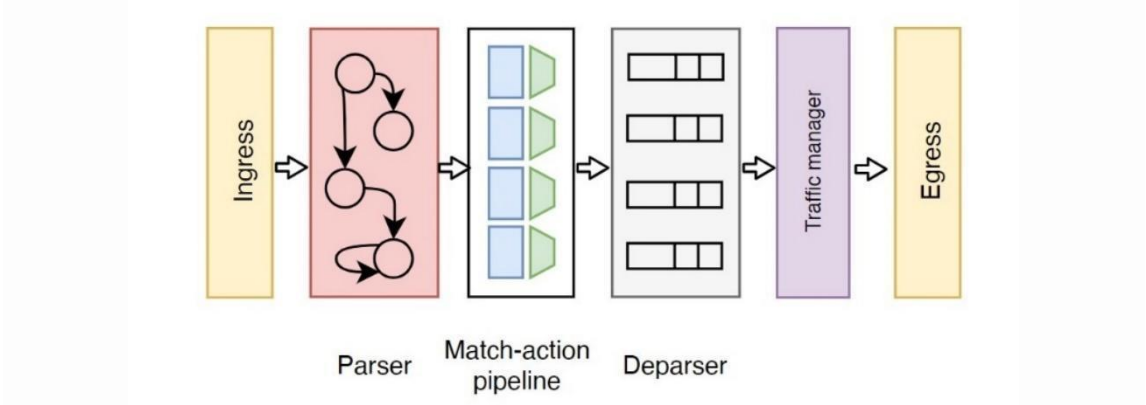


Figura 10- Arquitectura SimpleSume [53].

- ***V1model***, consiste num *parser* programável, um pipeline de *match-action* de entrada (*ingress*), um gestor de tráfego, um pipeline de *match-action* de saída (*egress*) e um *deparser* como podemos ver na Fig. 11. Esta arquitetura imita o pipeline de *match-action* de P414.

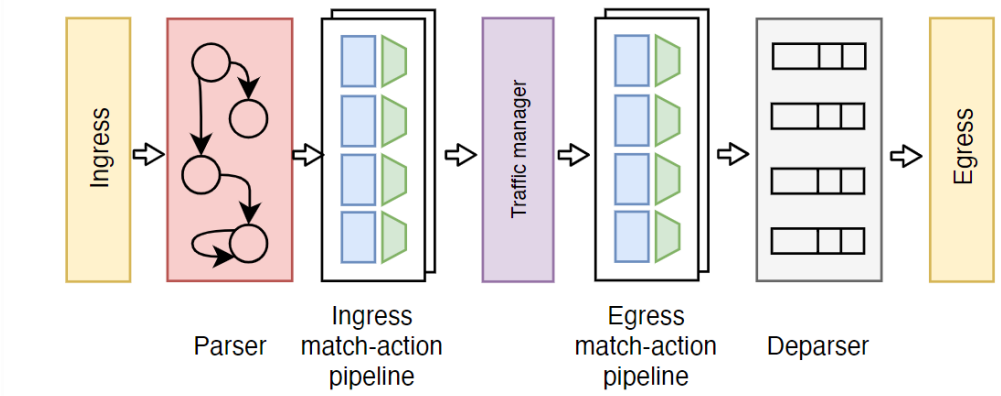


Figura 11 - Arquitetura V1model [53].

- ***Portable Switch Architecture*** (PSA), divide-se em pipeline de entrada e saída, em que cada um consiste em três partes programáveis: *parser*, múltiplos blocos de *match-action* e *deparser* conforme mostrado na Fig. 12. Neste caso, a arquitetura também define componentes configuráveis de função fixa. Além disso tem várias primitivas de processamento de pacotes específicas, tais como:
  - Envio de um pacote para uma porta *unicast*;
  - Descarte do pacote;
  - Envio do pacote para um grupo *multicast*;
  - Re-submissão de um pacote, movendo o pacote em processamento desde o fim do pipeline de entrada até ao início do pipeline de entrada para efeitos de *re-parsing* de pacotes;
  - Recirculação de um pacote, movendo o pacote em processamento do fim do pipeline de *egress* para o início do pipeline *ingress* para efeitos de transformação recursiva (como seja no caso de *tunneling*);
  - Clonagem de um pacote, duplicando o pacote atualmente processado. No caso de clonagem de *ingress* para *egress* (CI2E), é criada uma duplicação

do pacote de entrada no final do pipeline de *ingress*. Por sua vez a clonagem de *egress* para *egress* (CE2E) cria uma duplicação do pacote já processado pelo *deparser* no final do pipeline de *egress*. Nos dois casos, os pacotes clonados começam a ser processados no início do pipeline de saída, como se pode ver na Fig. 12. Esta clonagem pode ser útil para implementar aplicações mais poderosas como espelhamento (*mirroring*) ou telemetria.

Apesar do pipeline de *egress* providenciar capacidade adicional de *match-action*, como as implementações práticas partilham os componentes físicos (memória, ALU, etc) do *ingress*, a sua implementação tem um custo reduzido [56].

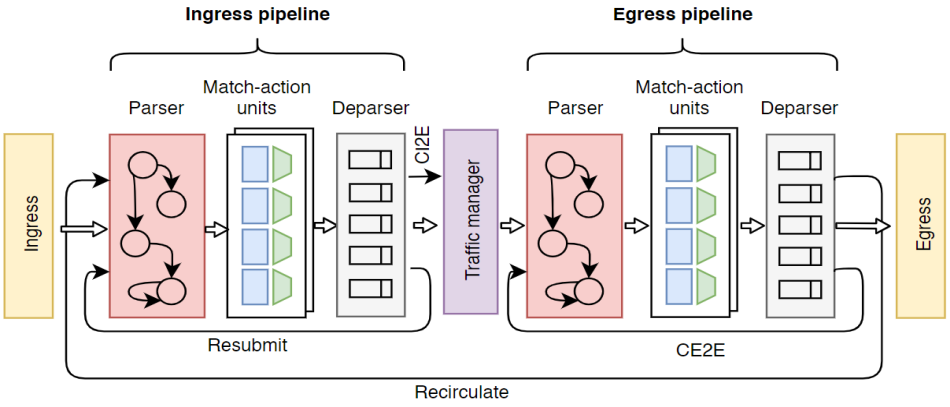
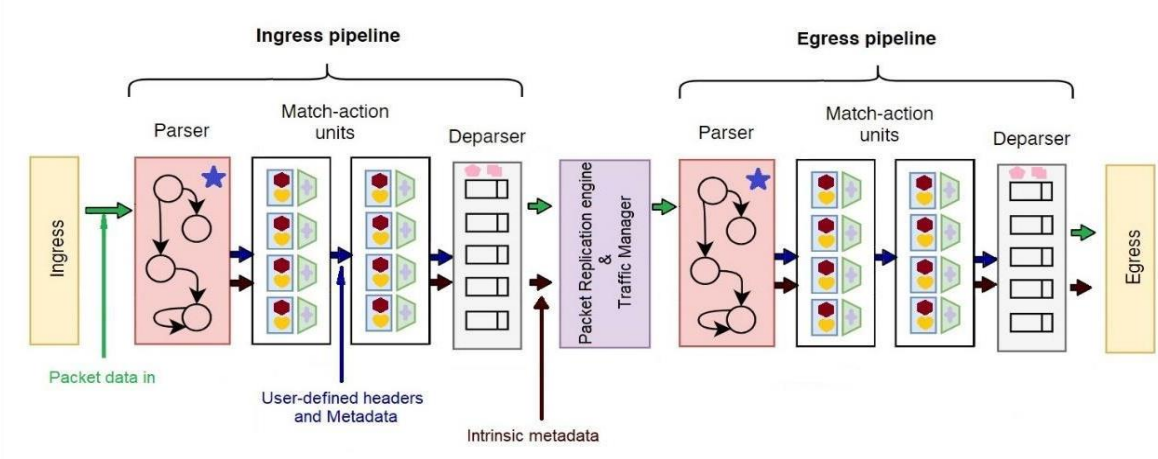


Figura 12 - Arquitetura PSA com partes de funções fixas e programáveis, e primitivas especiais de processamento de pacotes [53].

- ***Tofino Native Architecture*** (TNA), é uma arquitetura P416 proprietária, projetada para *switches ASICs Intel Tofino* que são dispositivos de muito alto desempenho e relativamente complexos. A unidade de programação básica desta arquitetura é um pacote chamado *Pipeline()* que se assemelha a uma versão estendida do *Portable Switch Architecture* (PSA) e é composto por 6 componentes programáveis (Fig. 13), de nível superior, distintos: *parser* de *ingress*, controlo de *match-action* de *ingress*, *desparser* de *ingress* e os seus homólogos de *egress*, mas com um motor de replicação de pacotes com características alargadas, assim como metadados intrínsecos e objetos externos específicos destes dispositivos.



O plano de controlo tem um papel importante no processamento de pacotes em tempo real através da gestão das MAT e dos componentes *stateful*, como são os contadores, registos, medidores (*meters)* ou objetos externos (*checksum*, unidades de computação *hash*, geradores de números aleatórios, etc). Estes objetos são elementos-chave para permitir medições de tráfego dentro do plano de dados. Particularmente, os registos (*registers*) podem ajudar a desacoplar estatísticas de tráfego das regras de encaminhamento e, consequentemente, melhorar o gerenciamento de memória. Enquanto o pipeline de processamento apenas permite manipulação dos cabeçalhos estes objetos podem também operar no *payload* do pacote. A comunicação entre este plano e o DP é proporcionada através de uma API como esquematizado na Fig. 14.

Figura 13 - Tofino Native Architecture [53].

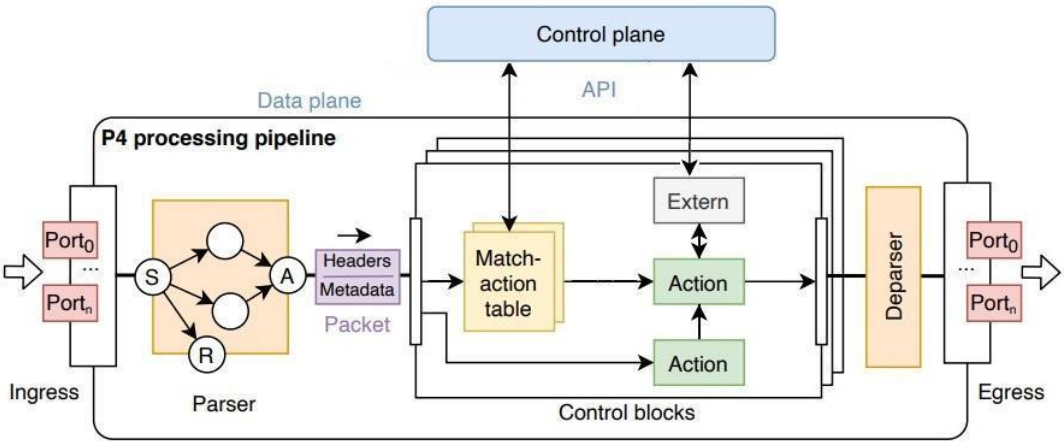


Figura 14 - Processamento de pipeline de P4 simplificado [57].

P4Runtime

O P4Runtime API é uma API independente de *target* e de programa e foi padronizada pelo P4 *Language Consortium*. O seu maior benefício é permitir ao controlador controlar qualquer plano de dados independentemente de ser construído a partir de ASIC fixo ou programável, FPGA ou NPU. Além disso, é independente das funcionalidades e protocolo que o plano de dados suporta, pelo que a mesma API pode ser usada para controlar um grande número de dispositivos [33].

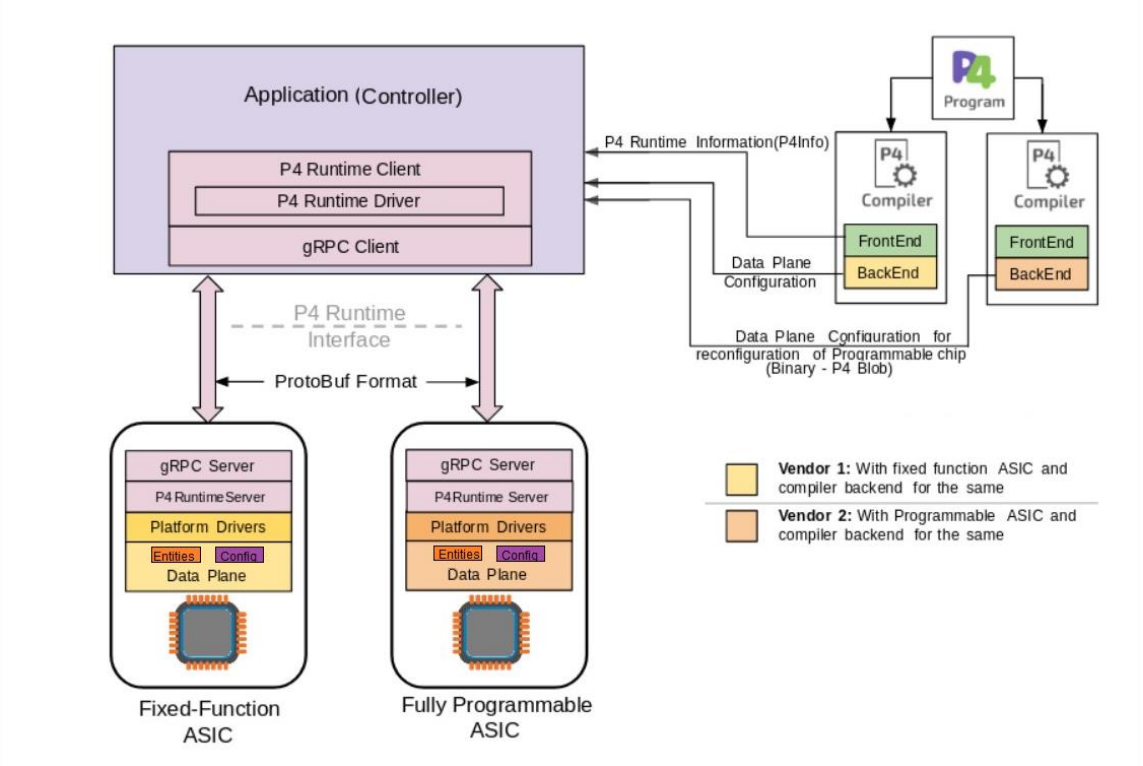


Figura 15 - Fluxo de P4Runtime entre o controladores e dois targets diferentes [56].

Conforme mostrado na Fig. 15, o P4Runtime usa gRPC (RPC - Remote Procedure Call) [59] para a comunicação entre o plano de controlo e os *targets* P4. Esta API suporta o acesso a objetos P4 (por exemplo, em MATs e *externs*), gestão de sessão (controladores de servidor/cliente), controlo de acesso baseado em funções, mecanismo de P*acket_in*/*Packet_out* para receber e enviar pacotes através do CP. É importante que sejam seguidas as boas práticas de segurança para proteger quer este canal comunicação quer os intervenientes (servidor e cliente de R4Runtime). Desta forma deve ser considerado o uso de TLS (*Transport Layer Security*) para autenticar e encriptar o canal gRPC e/ou autenticação mútua com certificados, para evitar ataques de *man-in-the-middle*, assim como se deve considerar a autenticação de conexões de entrada e controlo de acessos para o servidor P4Runtime, para prevenir o controlo malicioso deste servidor.

A Biblioteca PI, que faz parte dos *targets* P4, é a implementação de referência do servidor gRPC P4Runtime em C [60]. Implementa a funcionalidade genérica para objetos P4 internos, mas pode ser estendida através de objetos de configuração específicos de arquitetura (conjunto de declarações que descreve as partes programáveis de um dispositivo) ou *target* [61]. A API P4Runtime está disponível para controladores comuns como o OpenDaylight (através de plugin) ou ONOS.

Dispositivos de rede Programáveis

O P4 está disponível para uma variedade de dispositivos de rede programáveis. Com base em CPUs comerciais de uso genérico (COTS - *Commercially Off-The-Shelf*), estes sistemas utilizam implementações de *software* para fornecer funcionalidades arbitrárias. Embora isto, em geral, tenha um custo de desempenho, no que diz respeito à funcionalidade e complexidade, proporciona flexibilidade. O poder de processamento do

CPU limita a taxa de transferência nestes sistemas, e além disso, interrupções e efeitos de cache podem ter impacto na latência e na variação no atraso de pacotes (*jitter*).

Em projetos de *hardware* a funcionalidade do plano de dados pode ser implementada em circuito integrado específico de aplicação ou ASIC (*Application-specific Integrated Circuit*), um arranjo de portas programável em campo (FPGA - *Field Programmable Gate Array*) ou um processador de rede (NPU - *Network Processing Unit*). Essas plataformas geralmente oferecem alto desempenho devido a componentes de *hardware* dedicados e especializados, como chips de memória endereçável de conteúdo ternário (TCAM) para obter um processamento eficiente de pacotes. Já no caso do uso de dispositivos de *software*, o plano de dados executa toda a lógica de processamento numa unidade central de processamento (CPU - *Central Processing Unit*) comercial de uso genérico, usando estruturas de dados e algoritmos otimizados. Não obstante, a distinção entre planos de dados de *hardware* e *software* pode não ser muito clara. Por exemplo, um dispositivo baseado em *hardware* ainda pode invocar uma CPU COTS para executar funções que não são suportadas nativamente no *hardware* subjacente ou não requerem alto desempenho. Da mesma forma, os dispositivos modernos baseados em *software* contam com a assistência de recursos de *hardware* específicos de domínio por motivos de eficiência, como *Data Direct I/O* (DDIO), *Receive Side Scaling* (RSS) e cada vez mais transferências de SmartNIC para executar a lógica de processamento de pacotes parcialmente ou totalmente no *hardware* [62].

Apesar de existirem várias plataformas disponíveis para os *targets* programáveis na *framework* de SDN, as que se destacam, por um lado por serem os alvos mais utilizados na investigação de soluções, e por outro, pelo número de produtos comerciais disponíveis ou em desenvolvimento, são [63]:

- **CPU COTS** (*Commercially Off-The-Shelf CPU*) ou de uso não específico - Estes sistemas utilizam implementações de *software* para fornecer funcionalidades arbitrárias. São flexíveis no que diz respeito à funcionalidade e complexidade, mas o poder de processamento do CPU limita a taxa de transferência nestes sistemas, e além disso, interrupções e efeitos de cache podem ter impacto na latência e na variação no atraso de pacotes (*jitter*).
- **NPU** (*Network Processing Unit*) - É uma plataforma com vários núcleos otimizados para processamento de pacotes. Os dispositivos com esta arquitetura geralmente contêm vários blocos de *hardware* funcionais diferentes. Alguns desses blocos são dedicados a operações específicas de rede, como balanceamento de carga, criptografia ou pesquisas de tabela. Outros são dedicados a componentes programáveis que geralmente são usados para implementar novos protocolos de rede e/ou operações sobre pacotes. Esta arquitetura tanto pode ser usada em *switches* como em placas de interface de rede (NIC - *Network Interface Card*). Por exemplo, NICs programáveis podem ser equipados com NPU, que devido à sua arquitetura otimizada proporcionam uma melhoria no desempenho de taxa de transferência (*throughput*) e uma latência consistentemente baixa comparativamente com os dispositivos de CPU COTS, no entanto a flexibilidade é reduzida. Contudo, como os NPUs são *hardware* especializado estão disponíveis em menos formas do que os NICs de função fixa.
- **FPGA** (*Field-programmable Gate Arrays*) – Estes sistemas fornecem através de programação, conseguida com linguagens de descrição de *hardware* (HDL - *Hardware Description Languages*) – uma funcionalidade quase arbitrária. Limitadas apenas por restrições básicas de *hardware*, tais como recursos de

memória e limitações de tempo, as FPGAs muitas vezes superam as plataformas anteriores no que diz respeito a *throughput*, *jitter* e latência. Apesar de ser uma solução altamente flexível, a programação de FPGAs requer conhecimento especializado de *hardware* e a implementação de algoritmos HDL na rede é uma tarefa morosa.

- **ASIC** (*Application-Specific Integrated Circuits*) – São circuitos integrados específicos da aplicação, que contrariamente aos apresentados anteriormente são construídos com um propósito, mas têm um conjunto de instruções limitado. Através de otimizações conseguidas através de um alto grau de paralelismo, destacam-se no que diz respeito ao *throughput*, latência e *jitter* do processamento de pacotes, o que significa que são muito eficientes. No entanto, a expressividade limitada do conjunto de instruções da plataforma impõe limites à sua flexibilidade no que diz respeito à implementação de características.

Entre estas plataformas, torna-se evidente os compromissos existentes, por um lado, entre as propriedades de tempo de execução e os custos, e por outro, entre a flexibilidade e as restrições de recursos.

Vantagens do plano de dados programável baseado em P4

A flexibilidade introduzida pela programação de planos de dados introduz total flexibilidade no processamento de pacotes de rede, o que significa que, protocolos, algoritmos e funcionalidades podem ser adicionadas, modificadas ou removidas pelos operadores ou utilizadores de rede. Além disso, o facto de poder incluir no código apenas os componentes necessários para um determinado caso de utilização, diminui a complexidade e melhora a segurança e a eficiência em comparação com os dispositivos multiusos. A programação de plano de dados, em conjunto com plataformas de *hardware* adequadas, permite que designers de equipamentos de rede e até mesmo utilizadores desenhem aplicações únicas ou experimentem novos protocolos, deixando de depender de fornecedores de ASICs especializados de processamento de pacotes para implementar algoritmos personalizados. Mais ainda, novos algoritmos podem ser programados e implantados em questão de dias, em comparação com os longos períodos de desenvolvimento de novas soluções baseadas em hardware. A programação de planos de dados também é benéfica para os fabricantes de equipamentos de rede, permitindo-lhes criar facilmente produtos diferenciados, apesar de usarem o mesmo processamento de pacotes ASIC, com a possibilidade de manterem o produto proprietário sem necessitarem de partilhar com potenciais concorrentes que usem os mesmos ASICs, diferenciando-se assim deles.

Apesar das linguagens de programação usadas na programação do plano de dados ainda não terem atingido o grau de portabilidade atingido pelas linguagens de programação para fins gerais, o desenvolvimento de algoritmos de plano de dados numa linguagem de alto nível como o P4, tem o potencial de tornar os sistemas de telecomunicações significativamente mais independentes dos dispositivos subjacentes. Além disso, apesar de não o requerer, como dito acima, a programação de planos de dados incentiva a total transparência. No caso de partilha do código fonte, todas as definições para protocolos e comportamentos podem ser vistas, analisadas e fundamentadas, de modo que este beneficie do desenvolvimento e revisão da comunidade. Mais ainda, os utilizadores podem escolher *hardware* com custos menos elevados que sejam adequados aos seus propósitos, sobre os quais executam os algoritmos pretendidos [53].

Segundo [11] o valor da SDN está na capacidade para garantir a aplicação de políticas coerentes e melhor escalabilidade devido ao seu gerenciamento centralizado e programabilidade de rede. As novas soluções de segurança devem tirar partido da riqueza de informações disponíveis - relacionadas com a utilização da rede – na SDN, como aquelas obtidas pelo uso de P4, para aprimorar a aplicação de políticas de segurança, deteção e mitigação de anomalias.

É necessário desenvolver soluções capazes de detetar, prevenir e terminar os ataques que colocam em causa o funcionamento das redes. Assim o processamento de ataques de rede geralmente inclui três etapas [64]:

- Identificar os campos relevantes do cabeçalho do pacote e obter estatísticas sobre as características do tráfego - Este passo geralmente requer o conhecimento do máximo de informação possível.
- Identificar potenciais ameaças com base no valor da funcionalidade - Este passo geralmente requer alguns valores de parâmetro como base para a decisão, tais como a utilização atual dos recursos do equipamento, os limiares estabelecidos antecipadamente, e os parâmetros obtidos através da aprendizagem automática.
- Mitigar os ataques - As medidas incluem descartar o pacote, o equilíbrio de carga, a limitação da velocidade ou apenas o aviso de outros dispositivos enviando sinais ao controlador.

Em comparação com o OpenFlow, o plano de dados programável tem três vantagens fundamentais ao lidar com ataques de rede: visibilidade por pacote, escalabilidade e capacidade de processamento de alta velocidade.

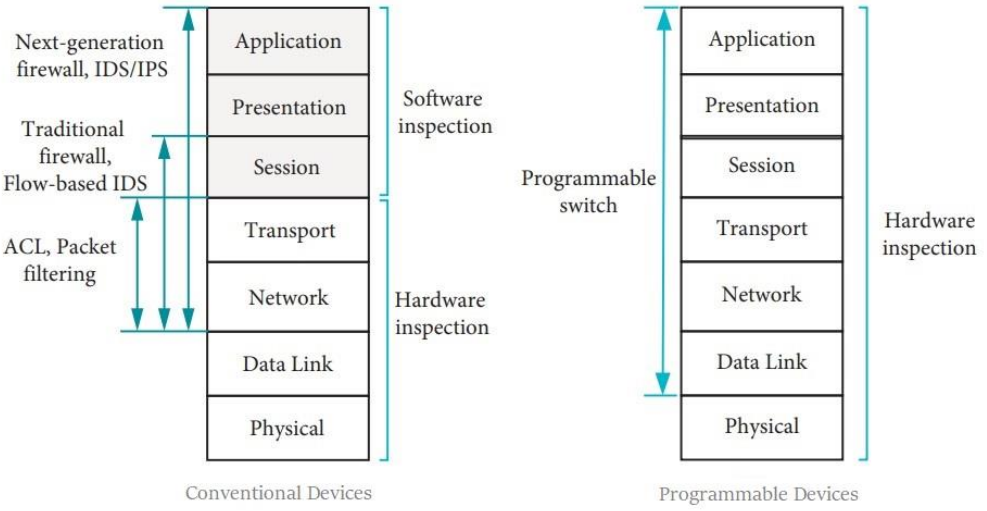


Figura 16 - Modelo OSI para aplicações de segurança. Adaptado de [64].

A visibilidade de cada pacote significa que os algoritmos de deteção de ataques podem ser desenvolvidos no *target* e aplicados a cada pacote em vez de usar amostragem ou rastreamento agregado no *software* do controlador. Tal como mostrado na Fig. 16, nas aplicações de segurança do modelo tradicional, as medidas de defesa possíveis de implementar ao nível do *hardware* incluem filtros de pacotes e ACL, enquanto *firewalls* de nível superior e estratégias de inspeção profunda de pacotes precisam de ser implementadas por *software* [65]. Esta é uma das razões pela qual os *switches* tradicionais

têm de enviar amostras de pacotes ao controlador. Pelo contrário, o *switch* programável tanto pode processar os campos de protocolo de cada camada ao nível do *hardware*, como cumprir os requisitos de inspeção profunda de pacotes, efetuar modificações dos campos de protocolo, ou suportar protocolos personalizados ou estratégias de defesa complexas.

Já a escalabilidade, potenciada pela programabilidade, significa que, uma vez que a defesa está localizada diretamente no *switch*, esta expandir-se-á com a velocidade e crescimento da rede, superando o problema de estrangulamento do controlador centralizado. No contexto das questões de segurança, o conhecimento a nível da rede fornecidos pelo controlador não trará benefícios adicionais para tarefas que apenas dependam do estado local. Contrariamente, a participação explícita do controlador em cada processo *stateful* irá causar comunicações de controlo adicionais que podem causar sobrecargas que levem ao seu mau funcionamento. O P4 fornece contadores de memória de estado (*counters*), *meters* e *registers*, que são usados para manter o estado entre vários pacotes do fluxo, permitindo que a maioria das medidas de defesa sejam completadas pelo plano de dados. Isto significa que o controlador apenas recebe e mantem um pequeno número de estatísticas, o que pode contribuir para a redução da complexidade da rede e garantir a escalabilidade da estratégia de defesa.

E por fim a capacidade de processamento de alta velocidade, significa que muitas tarefas podem ser feitas à velocidade de linha. O plano de dados programável pode executar políticas locais, processar pacotes à taxa de linha, responder rapidamente aos comportamentos de ataque, e satisfazer os requisitos das aplicações em tempo real. Uma vez detetado um ataque, permite que sejam imediatamente tomadas medidas no dispositivo de rede para mitigá-lo, sem introduzir atrasos pelo tempo de ida e volta da comunicação com o controlador, doutra forma necessária. Além disso, as aplicações com processamento complexo podem ser executadas a baixo custo e de forma distribuída no plano de dados. Algumas das funções originalmente implementadas pelo *software* do controlador podem ser transferidas para o plano de dados, que em vez de lhe transmitir os dados em bruto transmitirá somente os resultados da execução destas. Isto pode acelerar a execução dessas funções bem como reduzir consideravelmente a carga de processamento do controlador [66], uma vez que para tomar decisões relativas à mitigação do ataque este teria de recolher informações de um grande número de pacotes, o que aumentaria o *overhead* de comunicação assim como o consumo dos seus recursos computacionais. Esta capacidade de processamento de alta velocidade é especialmente importante na resolução de problemas de segurança.

Relativamente às vantagens de programação do plano de dados utilizando o P4 podem ser sumariadas conforme [67] e [50], da seguinte forma:

- Flexibilidade: O P4 torna expressáveis como programas muitas políticas de encaminhamento de pacotes, em contraste com os dispositivos de rede tradicionais, que expõem encaminhamento de função fixa.
- Expressividade: O P4 pode expressar algoritmos de processamento de pacotes sofisticados e independentes de *hardware* utilizando operações exclusivamente de uso geral e pesquisas de tabela. Estes programas são reutilizáveis em alvos de *hardware* que implementam as mesmas arquiteturas (assumindo que existem recursos suficientes).
- Mapeamento e gestão de recursos: Os programas P4 descrevem os recursos de armazenamento de forma abstrata (por exemplo, endereço de origem IPv4), enquanto os compiladores mapeiam esses campos definidos pelo utilizador para

recursos de *hardware* disponíveis e gerem detalhes de baixo nível, como alocação e agendamento.

- Engenharia de *software*: Os programas P4 proporcionam benefícios importantes, tais como verificação de tipo, ocultação de informações e reutilização de *software*.
- Bibliotecas de componentes: As bibliotecas de componentes fornecidas pelos fabricantes podem ser utilizadas para envolver funções específicas de *hardware* em construções de alto nível reutilizáveis P4.
- Desacoplamento de *hardware* e evolução de *software*: Os fabricantes-alvo podem usar arquiteturas abstratas para dissociar ainda mais a evolução de detalhes arquitetónicos de baixo nível do processamento de alto nível.
- Resolução de problemas (*Debugging*): Os fabricantes podem fornecer modelos de *software* de uma arquitetura para ajudar no desenvolvimento e resolução de problemas de programas P4.

As vantagens do plano de dados programável aliadas às caraterísticas da linguagem P4 acima referidas, bem como a simplicidade da sintaxe, o seu modelo de funcionamento, posicionam-na como uma linguagem adequada para alcançar as vantagens da programabilidade do plano de dados face às questões de segurança. Um exemplo disto é a possibilidade de a partir do uso apropriado dos blocos de *parser* e *deparser* poderem ser definidos protocolos personalizados que atendem aos requisitos e restrições de muitos ambientes. Um exemplo do potencial de P4 para interromper o ecossistema de rede oferecendo novas funcionalidades é o *In-band Network Telemetry* (INT) [68] [50], um conceito fortemente utilizado que permite a transferência de metadados de rede através das redes utilizando o tráfego gerado pelos utilizadores. A utilização de estatísticas de rede INT é anexada a um pacote de aplicações ao atravessar a rede. Antes de chegar ao seu destino final, os metadados INT são retirados do pacote e entregues ao controlador ou a qualquer outro dispositivo enquanto o pacote permanece inalterado do ponto de vista das aplicações. Métodos baseados em esboços *(Sketch-based)*, por outro lado, tentam manter uma estimativa do estado real utilizando contadores internos e matrizes de hash [69]. Mas as possibilidades que o P4 dá não se restringem a este tipo de soluções. Tarefas que eram tradicionalmente executadas em *hosts* intermédios com funções específicas *(midleboxes*) ou dispositivos com custos elevados, podem através do P4 ser implementadas no plano de dados. Assim o P4 eleva a capacidade do plano de dados programável assistindo na implementação de aplicações de nível superior em domínios tão diferentes como monitorização, engenharia de tráfego ou segurança de rede [39], sendo o último domínio o foco deste trabalho.

Garantir a segurança de uma rede de computadores é um desafio para qualquer organização. Nesse sentido, como uma abordagem de gestão, a *framework* de SDN reforça a aplicação das políticas de segurança necessárias para o conseguir. A utilização da linguagem de programação P4 nas redes SDN pode ajudar ainda mais a impor políticas ou detetar anomalias na taxa de linha, com latência praticamente nula. Segundo [64] os esforços feitos no desenvolvimento de soluções para a segurança de rede são muitos e abrangem áreas como o controlo de acesso, a privacidade e encriptação, a disponibilidade, e a defesa integrada conforme mostrado na Fig. 17.

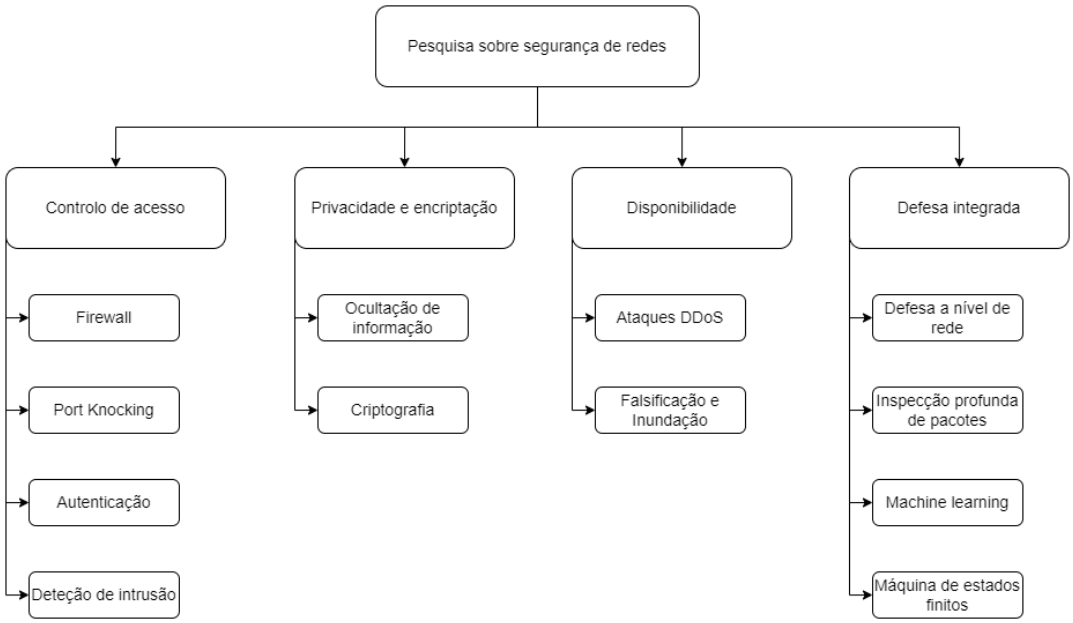


Figura 17 - Mapa de investigação sobre segurança de rede baseado no plano de dados programável P4. Adaptado de [64].

Assim, podemos ver que a linguagem P4 também tem um grande potencial na construção de uma grande variedade de aplicações no domínio da segurança. Por exemplo, os dispositivos programáveis podem ser incumbidos de responsabilidades de *firewall* e ser altamente flexíveis ao mesmo tempo.

Apesar dos benefícios, como a flexibilidade ou a expressividade, a linguagem de programação P4 também tem desafios. Não é uma linguagem Turing completa [67]. Apesar do seu objetivo ser o encaminhamento de dados, existem algumas tarefas do encaminhamento que não são possíveis de expressar através de P4 (como é o caso de *broadcast* ou *multicast*), pelo que por vezes é necessário invocar objetos externos (*externs*) para concretizar tarefas impossíveis de outra forma. Estes objetos externos são dependentes da plataforma-alvo P4, não existindo uma uniformização do conjunto destes objetos. Para mais, não há construção de iteração em P4, sendo que só a máquina de estado *parser* pode criar *loops*. Também não tem suporte para funções recursivas, consequentemente, o trabalho realizado por um programa P4 depende linearmente apenas do tamanho do cabeçalho do pacote. Todo o estado de um programa P4 é criado quando um pacote é recebido e destruído quando o processamento está completo, o que significa que análises relativamente a fluxos de pacotes só são conseguidas com suporte externo. Mais ainda, não existe alocação dinâmica de memória e não suporta ponteiros ou referências. E tratando-se de uma linguagem, há que ter em consideração possíveis bugs dos algoritmos construídos, pelo que é crucial a identificação nas fases iniciais do desenvolvimento destes, para prevenir eventuais falhas de rede ou comportamentos inesperados.

Apesar destas limitações, o P4 é uma linguagem extraordinariamente útil para a programação do plano de dados. As vantagens que oferece fomenta o interesse por parte da comunidade que investiga esta área, tornando-a a linguagem de programação padrão *de facto* para descrever o encaminhamento de pacotes em dispositivos de redes programáveis [49] [70] [71].

Assim, e tendo em conta tudo que foi apresentado até ao momento, no contexto de SDN, o P4 é uma linguagem de programação adequada para a implementação de mecanismos de defesa de ataques DDoS, nomeadamente de ataques de *SYN Flood*. Através dela, poderá ser implementada uma solução que alivie não só o problema de saturação do plano de dados, mas também do plano de controlo. Ao utilizar os objetos *registers* desta linguagem para obter a informação necessária à tomada de decisão por parte do plano de dados, há um alívio do controlador. Através destes *registers* poderão ser implementadas soluções de entrada de tabela focadas no plano de dados, nomeadamente análise e recolha de estatísticas, de gestão de falhas em tabelas, de tabela de fluxo ou de largura de banda, que tornarão os dispositivos de rede mais inteligentes e permitirão uma resposta imediata a ataques.

3.6 Sumário

Neste capítulo foi introduzida a estrutura base da arquitetura SDN, nomeadamente os seus três planos – plano de dados, plano de controlo e plano de aplicação – bem como as interfaces presentes na solução e as principais características que a definem. Foram ainda expostos os benefícios e as limitações da arquitetura SDN, com destaque para o desafio da segurança sobre o qual se foca este trabalho.

Relativamente à segurança, foram descritos os pontos-alvo, as ameaças e vulnerabilidades a que a SDN está sujeita. Foi também identificado o tipo de soluções que podem ser tomadas na investigação de melhorias de segurança, nomeadamente, em aplicações ou no design da arquitetura desta tecnologia. Foram ainda identificadas as categorias de ataque DDoS que afetam a SDN e os mecanismos de defesa que podem ser usados no desenvolvimento de soluções de defesa, prevenção e mitigação a estes ataques.

Foi apresentada a linguagem P4, assim como os blocos do modelo PISA, na qual esta se baseia: *parser*, pipeline *match-action* e *deparser*. Adicionalmente foi evidenciado o que o distingue do seu predecessor, o Openflow. Sobre o qual também foi revisto o seu funcionamento para contextualização da necessidade de uma solução como o P4.

Ainda relacionado com P4, foi discutido o P4Runtime - a API de comunicação entre o plano de controlo e o plano de dados, que proporciona o controlo do DP por parte do controlador, independentemente do tipo de targets subjacentes (fixos ou programáveis). Além disso, foram também apresentadas as plataformas de dispositivos programáveis que mais se destacam e sobre os quais o P4 pode ter um impacto significativo na sua programação.

Finalmente, foram apresentadas as vantagens do plano de dados programável que aliadas às caraterísticas da linguagem P4 podem contribuir para o desenvolvimento de soluções mais eficientes de resposta aos problemas identificados.

Capítulo 4
Solução proposta: DPSynFloodBlock

4.1 Experimentação inicial

Para desenvolver uma solução de segurança com foco no plano de dados foi necessário compreender o funcionamento das redes SDN e da linguagem P4 do ponto de vista prático. Assim, foram efetuados estudos experimentais para compreender, por um lado, o funcionamento das redes SDN, utilizando a ferramenta Mininet, e por outro, a linguagem P4.

Mininet

Apesar de existirem várias opções disponíveis como emuladores e simuladores, optou-se pela ferramenta *Mininet* [74]. Trata-se de um emulador de redes que permite desenvolver e testar redes de grande escala num único sistema (com recursos limitados).

O *Mininet* é um emulador de rede do tipo CBE (*Container-Based Emulation*), com o processo de emulação mais rápido e escalável devido à virtualização ao nível de processos, onde os recursos, como estrutura de dados, sistemas de ficheiros e *kernel*, são partilhados. Com a ajuda deste emulador, é possível realizar experiências de redes com centenas de comutadores e/ou *hosts*, com um custo reduzido, onde são permitidas alterações de configuração de forma rápida. No contexto de SDN permite emular comutadores com suporte OpenFlow (Open vSwitch), comutadores de *software* P4 (bmv2 - *behavioral model version 2*), assim como a interação com os controladores conectados a estes (locais ou remotos) e interagir com elementos externos baseados em *hardware*. Mais ainda, permite emular os links de ligação entre duas interfaces virtuais.

Esta ferramenta dispõe de uma interface de linha de comandos (CLI - *Command Line Interface*) para comunicar e controlar os dispositivos virtuais e a rede. Também permite a comunicação entre dispositivos através por exemplo do uso do comando *ping* ou de ferramentas específicas, para testar a conectividade entre eles. Adicionalmente disponibiliza uma API para a linguagem de programação *Python* [75] tornando possível criar ou usar *scripts* em *Python*, para simular na ferramenta as redes pretendidas, bem como simular tráfego de rede entre os *hosts*.

Apresenta algumas vantagens relativamente a outros emuladores, tais como melhor velocidade de arranque, maior escalabilidade, maior largura de banda e ainda permite projetar, implementar e testar topologias de rede complexas antes da sua implementação na rede real. No entanto também tem algumas limitações, como a utilização de um único *kernel* Linux para todos os *hosts* impedindo a execução de *software* que dependa de outros *kernels*. Além do mais, isola por defeito a topologia da rede emulada da rede local e da Internet e está limitado à utilização do CPU e largura de banda de um único servidor [27].

A integração de outras ferramentas tais como o *Iperf* [76], ou o *Wireshark* [77] entre outras, permite enviar e monitorizar pacotes entre os *hosts* e assim compreender as interações entre os vários elementos da rede.

Explorou-se os conceitos básicos da ferramenta, tal como a criação dos vários elementos da rede através da linha de comando, seguindo os conteúdos de [78], e [79]. Foi também estudada a ferramenta *Miniedit* [80], uma interface gráfica disponibilizada pelo projeto *Mininet* seguindo o Labaratório 1 do conteúdo [81].

Linguagem P4

Com a intenção de aprender a linguagem P416 [49] foram seguidos os exercícios [81], [82] disponibilizados pela *University of South Carolina*. Estes exercícios permitiram a compreensão das instruções necessárias na construção dos vários elementos que fazem parte da estrutura de um programa P4 com vista a definir o comportamento de um *switch* programável. Estes tutoriais basearam-se no fluxo de trabalho (*workload*), conforme mostrado na Fig. 18, para a programação de um *switch* por *software* BMv2, com a arquitetura V1model.

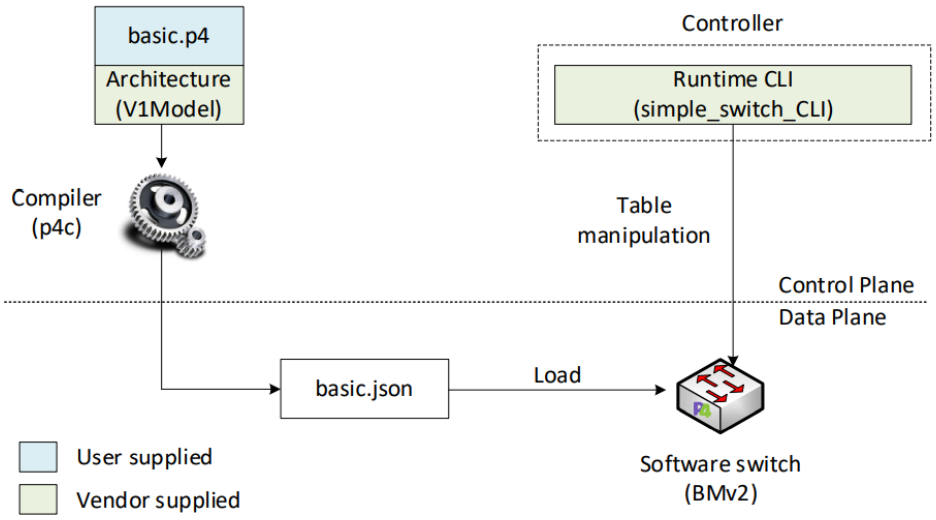


Figura 18 - - Fluxo de trabalho dos exercícios usados em [82].

Particularmente os exercícios de [82] e a VM associada a estes, permitiram a familiarização com a estrutura e componentes de um programa P4, entre as quais, a definição dos cabeçalhos (*header)*, das tabelas (*table*), das ações (*action*), *checksum*, etc, e de como mapear os componentes do programa P4 no modelo PISA.

Já os exercícios de [81] e a VM correspondente, também foram úteis para entender o papel dos componentes *stateful* para uma verdadeira programabilidade dos *switches*. Mais especificamente, para compreender os componentes *externs*, que são dependentes do modelo de arquitetura P4 específico do *switch.*

Permitiram perceber como se pode recorrer aos contadores (*counter*) para implementar algumas tarefas, tais como recolher estatísticas de fluxos, impor políticas de Qualidade de Serviço (QoS) e implementar funcionalidades de segurança, como sejam detetar e bloquear ataques de Negação de Serviço (DoS) – de salientar que os *switches* apenas escrevem nestes contadores, sendo do controlador a tarefa de os ler e decidir as próximas ações baseadas nos mesmos. Também permitiram compreender a diferença entre contadores diretos e indiretos. Assim, os de tipo direto são aqueles que estão diretamente associados a uma MAT (por exemplo para contar quantas vezes uma regra em uma MAT

foi acedida), já os indiretos, são contadores independentes invocados por *externs* que podem ser associados a entradas ou grupos de entradas específicas numa MAT conforme mostrado na Fig. 19. No caso dos contadores diretos o número de índices do contador é igual ao tamanho da tabela, contrariamente aos contadores indiretos onde o mesmo índice (Idx.) pode aparecer em várias linhas da tabela.

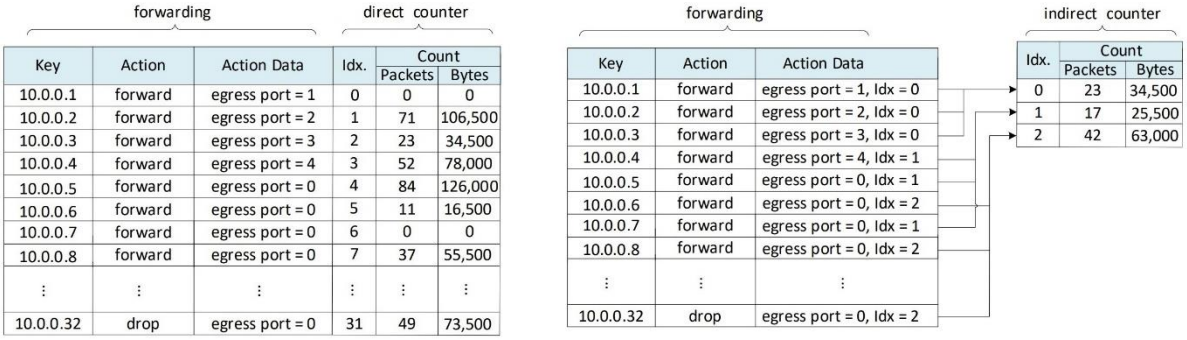


Figura 19 - Exemplos de duas MAT com contadores diretos (à esquerda) e indiretos (à direita). Adaptado de [81].

Similarmente, também levaram à compreensão de como se pode recorrer aos *meters*, para medir e marcar o débito de tráfego de entrada e para a implementação de políticas de Qualidade de Serviço (QoS). Os *meters* são definidos como mecanismos para caracterizar o tráfego num número predeterminado de estados, dois exemplos destes mecanismos são documentados em [83], [84]. Do ponto de vista prático estes objetos, poderão, por exemplo, ser úteis para separar o tráfego malicioso para um *blackhole*, ou aplicação de segurança especializada, para uma análise mais detalhada.

Também foi útil para entender como é possível através dos *registers* armazenar informação arbitrária que pode ser acedida durante o processamento dos múltiplos pacotes que atravessam o *switch.* Estes *registers* podem ser lidos e escritos quer pelo controlador, quer pelo *switch.* Quando lidos pelo *switch* podem ser úteis, por exemplo, para calcular o intervalo de tempo entre pacotes pertencentes ao mesmo fluxo. Este objeto também poderá ser usado na análise de ataques de DDoS do tipo *SYN Flooding* - diretamente no *switch* sem recorrer ao controlador - para guardar informação relacionada com os pacotes SYN anteriormente recebidos, que chegaram ao *switch*, de forma a poder analisar se existiu um ACK após o envio do SYN/ACK. Ou ainda para analisar se determinado porto do switch está a receber ou não um fluxo anormal de pacotes com uma grande entropia de endereços IP.

O último exercício proposto em [81] proporciona uma demonstração como pode o *switch* pode notificar o controlador, enviando-lhe informação. No entanto este último terá de saber interpretar a informação que lhe é enviada. Por exemplo, este objeto permite alimentar o controlador com informação pertinente no suporte ao plano de aplicação.

Esta experimentação permitiu por um lado compreender melhor a semântica da linguagem P416, mas também contextualizar a forma como esta linguagem pode ser útil na construção de soluções mais eficientes em dispositivos de rede programáveis, permitindo o desenvolvimento da solução apresentada em seguida.

4.2 Ambiente de desenvolvimento

Para o desenvolvimento da solução aqui proposta, disponível em [85], recorreu-se à imagem virtual Ubuntu 20.04 para VirtualBox disponibilizada em [86], que já tem instaladas e compiladas ferramentas de código aberto P4. A partir desta imagem foi então desenvolvido o ambiente de teste usado neste trabalho, que passou por usar uma *testbed* inteiramente implementada em software, recorrendo à ferramenta *Mininet*, com dispositivos de rede de *software*, concretamente *switches* BMv2, que apesar de terem algumas limitações já referidas neste trabalho, servem para o propósito desta investigação na criação de uma prova de conceito de resposta de primeira linha, no plano de dados a ataques DDoS *SYN Flood*.

Para realizar a comunicação entre os vários componentes das testbed foram definidas as MAT correspondentes, validadas com a ferramenta *ping* para verificar a conectividade básica entre vários nós da rede.

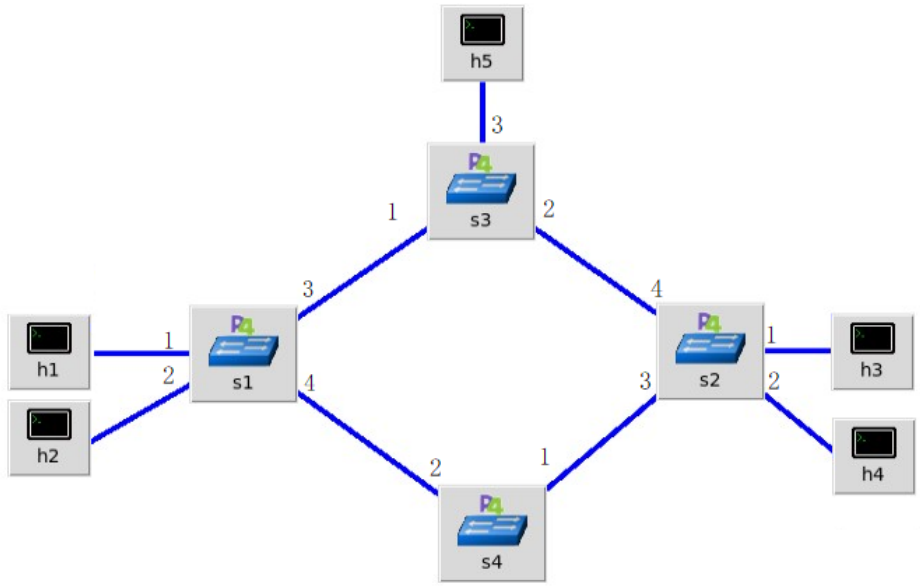


Figura 20 – *Testbed* usada para o desenvolvimento da solução DPSynFloodBlock

Para os testes ao algoritmo foram usadas as ferramentas *Scapy* [87], *hping* [88], *top* [89], W*ireshark* [76], *iperf* [77], *ping*, dois *scripts* em *python* adaptados de [90] para as particularidades dos testes, nomeadamente ddos_send.py e ddos_send_range.py, e três outros *scripts* também em *python*, escritos para simplificar e automatizar os testes (h_comando_generico_time_limit.py, h_comando_generico_time_limit_attack.py e h_comando_generico_packet_limit) . Na *testbed*, que poderá ser observada na Fig. 20, são usados quatro *switches* pertencentes a uma rede que pretendemos proteger.

4.3 Solução DPSynFloodBlock

Objectivos e estrutura

Pretendeu-se implementar uma solução no plano de dados com recurso a linguagem P4, que aliviasse os recursos do controlador, já que este é o ponto principal de

estrangulamento numa perspetiva da rede como um todo, bem como fosse uma solução que não alterasse o conteúdo do pacote. Nesse sentido, pretendeu-se que a solução fosse simples, sem dependências diretas entre os outros dispositivos - apesar de se poderem complementar - e que dependesse o mínimo possível da interação com o controlador, diminuindo o esforço de modo reativo do controlador, contribuindo para uma melhoria do desempenho deste.

A escolha de uma solução para ataques *DDoS Syn Flood* prendeu-se ao facto destes continuarem a representar uma grande parte dos ataques de DDoS, devido à simplicidade da sua implementação relativamente ao impacto que causam. Estes ataques usam o processo de estabelecimento de conexão do protocolo, conhecido como *three-way handshake*, para efetivar o ataque. No *three-way handshake* (Fig. 21), os pedidos são feitos através do envio de uma mensagem SYN ao servidor, ao qual este responde com mensagem de SYN ACK, a partir do qual este reserva os recursos necessários para a efetivação do estabelecimento da conexão, que será totalmente estabelecida quando o cliente enviar a mensagem ACK, de resposta à mensagem SYN/ACK do servidor. Apesar do servidor ter medidas de salvaguarda nas quais é feita a libertação dos recursos após um tempo determinado, um agente malicioso que envie muitos pedidos (por exemplo com recurso a uma *botnet* ou com *ip spoofing*) sem enviar a mensagem de ACK, fará com que todos os recursos do servidor (memória e processamento) sejam ocupados com os seus pedidos, impedindo o serviço ao tráfego legítimo.

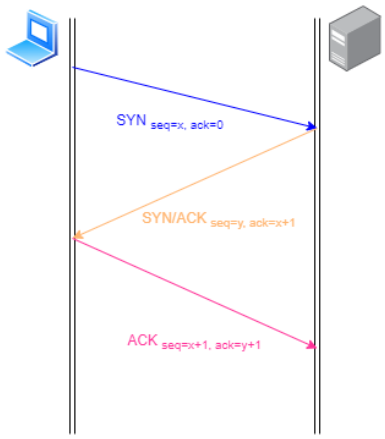


Figura 21 - Processo de estabelecimento de conexão do protocolo TCP .

A solução para implementação no plano de dados, para controle de ataques de DDoS *SYN Flood* desenvolvida, DPSynFloodBlock cujo fluxo de alto nível pode ser visto no Anexo A, tem como objetivo ser uma primeira linha de defesa, impedindo grande parte do tráfego malicioso. Tem como ideia principal, identificar um ataque de *SYN Flood* com recurso ao número de pedidos SYN e ACK, passados no switch caso seja identificado um ataque, o switch bloqueará o tráfego que chega ao porto onde foi identificado o tráfego malicioso, até que o ataque não se verifique mais. Nesse caso o porto deixará de estar bloqueado, mas os pedidos não serão passados imediatamente para o seu destino. Recorrendo às características do protocolo TCP [91], nomeadamente o seu comportamento para o caso de congestão, onde um *host*, não malicioso, que não recebe resposta após o intervalo de tempo RTO (*Retransmission Time Out*) – calculado a partir do RTT (R*etransmission Trip-Time*) - ao seu pedido inicial SYN, reenviará o pedido novamente e que no caso de também este segundo pedido não obter resposta uma vez mais, levará o *host* a reenviar um terceiro pedido que será enviado num intervalo de tempo compreendido entre RTO e 2xRTO conforme mostrado na Fig. 22. De acordo com o RFC

6298, se alguma ligação já tiver sido anteriormente estabelecida, RTO será calculado a partir do RTT (R*etransmission Trip-Time*) [92][93], senão RTO será igual a 1 segundo.

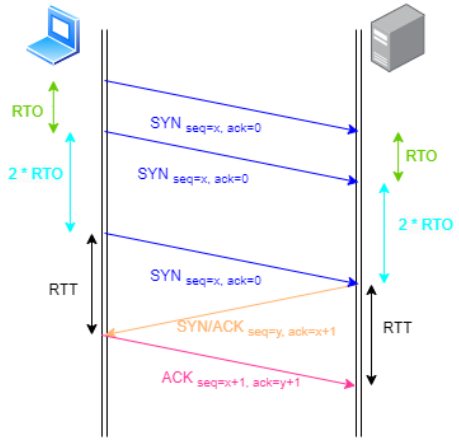


Figura 22 - RTO (Retransmission Time Out) e RTT (Retransmission Trip-Time).

Normalmente a monitorização das estatísticas de RTT estão disponíveis nos *hosts*. Mas, apesar do seu elevado custo associado à performance da rede, fazer a monitorização destas nos dispositivos intermédios à comunicação dá-nos informação valiosa relacionada não só com métricas de latência ou performance de rede, mas também informação sobre possíveis problemas de segurança na rede.

Assim, a análise da informação de RTT feita no *switch*, será usada como forma de prevenção na solução aqui descrita, e usada imediatamente após identificação de que o ataque já não se verifica. O *switch* apenas deixará passar um pedido SYN por parte do suplicante, ao fim da terceira tentativa, se esse pedido, chegar no intervalo esperado. Caso contrário é considerado tráfego malicioso e, portanto, é descartado (*drop*).

Do ponto de vista da estrutura da solução em P4 disponível em [85], esta tem quatro fases distintas: monitorização, deteção, mitigação e prevenção. É composta pela pipeline das tabelas *match-action* evidenciadas na Tabela VI e pelas ações, não diretamente associadas às tabelas *match-action*, mas complementares a estas, listadas na tabela VII.

Tabela VI - Tabelas *match-action* usadas na solução DPSynFloodBlock.

Tabela	**Chave**	**Método**	**Ações**	**Tamanho**	**Acção-por-defeito**
ipv4_lpm	hdr.ipv4.dstAddr	lpm	ipv4_forward; drop; NoAction	1024	drop
forwarding_arp	hdr.arp.tpa	exact	forward_arp; drop; NoAction	1024	drop
drop_table		exact	drop		
blacklist	hdr.ipv4.srcAddr	exact	drop_blacklist	1024	NoAction
port_blacklist	standard_metadata.ingress_port	exact	drop_blacklist	SWITCH_PORTS_NUMBER	NoAction
target_host	hdr.ipv4.dstAddr	exact	target_host_forward	PROTECTED_TARGET	NoAction
source_host	hdr.ipv4.srcAddr	exact	source_host_forward	PROTECTED_TARGET	NoAction
hostportind	hdr.ipv4.dstAddr standard_metadata.ingress_port	exact exact		SWITCH_PORTS_NUMBER * PROTECTED_TARGET	NoAction

Tabela VII – Ações complementares às tabelas *match-action*.

Ações complementares
Set_conn
Compute_flow_id
Get_interarrival_time
Set_payload_size
Set_eACK
Set_flowID
Set_key
do_update_syn_reg
do_update_synack_reg
do_update_ack_reg
get_min_num

Monitorização e deteção

Para a fase de monitorização, foram usados registos que armazenam o número total de pedidos SYN e o número total de mensagens ACK recebidas por *host* protegido, *flow_reg_syn* e *flow_reg_ack* respetivamente; assim como por par *host protegido – porto de ingresso do switch (*ou *host-port*, que é dado pelo valor *pair-num* da tabela *hostportind*), neste caso é usado o registo *host_3hand_reg*. Para a deteção é verificado se a diferença entre pacotes de pedido de ligação TCP, SYN, e a confirmação de estabelecimento dessa ligação, ACK, ultrapassam um valor predeterminado quer por *host* (dado por SYN_ACK_THRESHOLD) quer por *host-port* (dado por PORT_SYN_ACK_THRESHOLD) conforme mostrada na Fig. 23.

DPSynFloodBlock.p4

```
...
flow_reg_syn.read(value_aux_syn,(bit<32>)meta.target_host_index);
flow_reg_ack.read(value_aux_ack,(bit<32>)meta.target_host_index);

...
if (value_aux_syn - value_aux_ack > (bit<48>)SYN_ACK_Threshold)  {
    Alarm_ddos = 1;
    if ((aux_port[47:0] - aux_port[143:96]) > (bit<48>)PORT_SYN_ACK_Threshold){
        meta.alarm =1;
    }
}
...
```

Figura 23 - DPSynFloodBlock: Monitorização.

No caso afirmativo, será então considerado que está sobre ataque, pelo que a informação do *timestamp* de chegada do pacote a partir do qual é identificado o ataque, bem como o número de pacotes SYN e de ACK (por *host-port*) que chegaram até ao momento serão guardados no registo digest_reg para o par host-port. Também o número de novas origens

ainda não conhecidas do switch com pedidos SYN, será registada no mesmo registo. Além disso sempre que a diferença entre SYN e ACK é múltiplo de 1024, o controlador é informado sobre o ataque, dando a este a possibilidade de intervir através da atualização das tabelas *blacklist* (lista de endereços IP bloqueados) ou *port_blacklist* (lista de portos de entrada no switch bloqueados). Optou-se por enviar somente informação de ataque ao controlador, quando a diferença do número de pedidos SYN e da confirmação ACK, provenientes de origens desconhecidas atinge um valor predefinido (múltiplos de 1024), por forma a não assoberbar o controlador com informação. Após a primeira informação ao controlador só será reenviada nova informação se verificada a condição referida anteriormente e se passou determinado período (TIMESTAMP_5) desde que o ataque se verificou. Adicionalmente no caso de o ataque não se verificar após o mesmo período, o controlador é também informado disso mesmo. As mensagens enviadas ao controlador são definidas através do comando *digest(digest_num,meta.syn_flood_attack_digest)* cujo significado de d*igest_num* é mostrado na tabela VIII. A informação enviada é dada pelo conteúdo de *meta.syn_flood_attack_digest* definida pela estrutura *digest_t* mostrado na Fig. 24, onde *malicious_port_source* corresponde ao porto de entrada do dispositivo programável onde chegou o pacote malicioso; o *syn_ack_diff* corresponde à diferença entre o número de mensagens SYN e ACK desde o última atualização de *meta.syn_flood_attack_digest* para o porto correspondente; *malicious_attack_destination* corresponde ao endereço IP de destino do último pacote malicioso*;* e *number_of_syn_from_unknown_sources* corresponde ao número de pacotes provenientes de origens desconhecidas do dispositivo programável das quais chegaram pedidos SYN.

Tabela VIII – *Digest_nu*m - Comandos de envio de informação ao controlador.

Digest_num	**Significado**
1	Sob ataque, mais de 1024 mensagens recebidas de origens desconhecidas
3	Ataque não verificado no intervalo TIMESTAMP_5

```
struct digest_t {
    bit<9> malicious_port_source;
    bit<128> syn_ack_diff;
    bit<32> malicious_attack_destination;
    bit<48>number_of_syn_from_unknown_sources;
}

struct metadata {
    digest_t  syn_flood_attack_digest;
```

Figura 24 - Estrutura digest_t definida em DPSynFloodBlock

Mitigação e prevenção

No caso do pacote que chega ao *switch* não estar abrangido por qualquer lista de bloqueio então passar-se-á à fase de mitigação. Esta divide-se em dois subtipos, conforme o objetivo que queremos obter. O primeiro subtipo, ou solução 0, tem como objetivo principal proteger uma máquina/serviço quando esta está sob-ataque, de pacotes provenientes de fontes desconhecidas, e assim bloquear todos os pedidos destas fontes

que chegam a um porto do *switch* específico, com destino a um determinado *host* que queremos proteger; e o segundo, ou solução 1, corresponde ao tipo de mitigação onde se pretende bloquear todos os pacotes que chegam a determinado porto do switch, com destino à máquina/serviço protegido, independentemente da proveniência deste, ou seja mesmo no caso de ser de uma origem já com registo de ligações TCP bem sucedidas que chegaram a esse mesmo porto do switch. Estes subtipos são definidos nas tabelas *target_host* e *source_host*. Estas tabelas são complementares e funcionam em associação. O bloqueio será avaliado em intervalos de tempo pré-estabelecidos (TIMESTAMP_5) e no caso de o ataque não se verificar no intervalo anterior (ou seja, os valores dos pedidos SYN menos os pedidos ACK não ultrapassar o limite estabelecido), para origens desconhecidas passar-se-á à fase de prevenção (aplicável em ambas as soluções, 0 e 1), onde iremos encontrar o comportamento discutido anteriormente, onde é implementada a estratégia inspirada no comportamento do protocolo TCP na presença de congestão de rede, conforme demonstrado na Fig. 25. Este comportamento é também o que vai acontecer no caso da solução 0, na situação em que o *host* está sob ataque, mas quando um pacote for proveniente de uma origem para a qual já exista um registo de uma ligação TCP bem-sucedida. Assim, nesta fase de prevenção, e como pode ser observado em detalhe no diagrama de fluxo de DPSynFloodBlock apresentado no Anexo B, o primeiro pedido SYN para um *host* protegido é descartado e registado o *timestamp* da sua chegada, o número de sequência (*seq_num*) e número de vezes que o pedido chegou (neste caso uma vez) no registo $last\_timestamp\_syn\_reg_x$, para o identificador do fluxo dado pelo *hash* da informação composta pelo *boot_timestamp_reg + ((2*x) + (x-1))* - funciona como uma salvaguarda adicional de segurança, comummente referido como sal no contexto de criptografia - e do tuple *ipv4.srcAdd + ipv4.dstAdd + ipv4.srcPort + ipv4.dstPort*, como mostrado nas Tabelas IX e X. Também é atualizada a informação nos registos *host_3hang_reg* e *flow_reg_syn*. No caso de um pedido legítimo o *host* suplicante interpretará a falta de resposta como congestão, pelo que reenviará o pedido SYN - com o mesmo número de sequência - após o intervalo de tempo RTO (Retransmission Time Out), conforme o definido no ponto 3.8.1 do RFC 9293. Em caso de correspondência encontrada no registo *last_timestamp_syn_reg* a um primeiro SYN, a informação referente a este pacote é atualizada no mesmo registo, nomeadamente o timestamp de chegada deste último pedido, o número de vezes que o pedido chegou - neste caso duas - e o intervalo de tempo entre os dois pedidos.

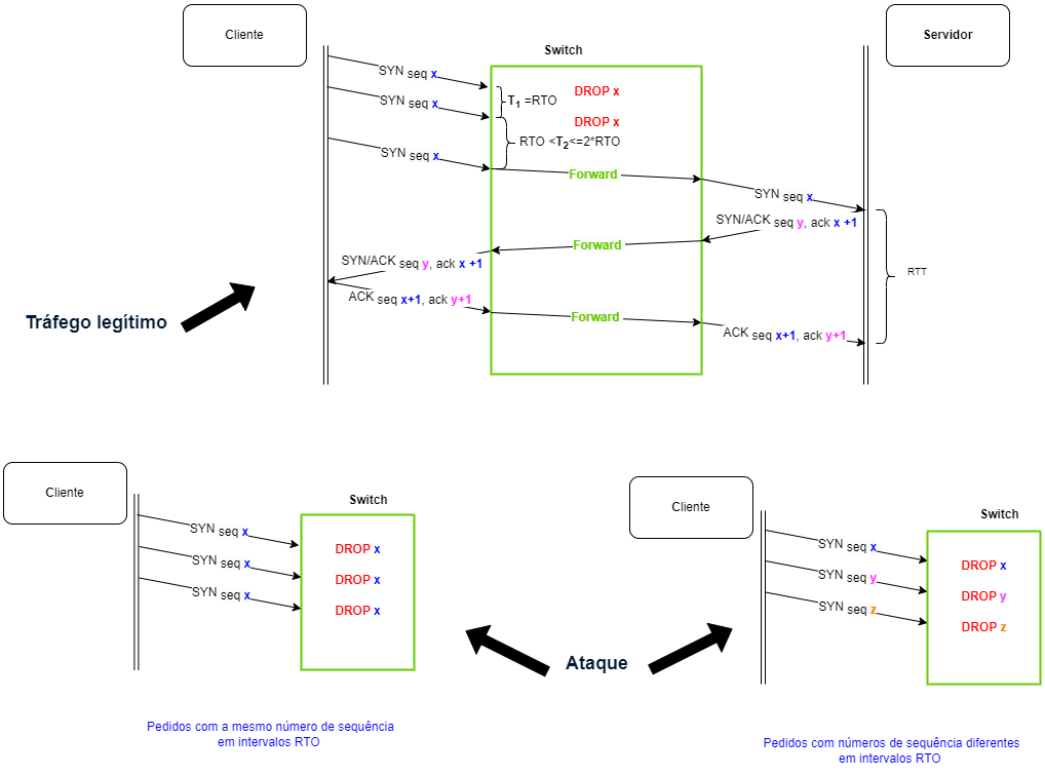


Figura 25- Cenários considerados na fase de mitigação da solução proposta.

Quando se trata de um *host* legítimo, também este pedido, a não ser respondido, levá-lo-á a enviar um terceiro pedido, após um intervalo de tempo compreendido entre RTO e 2xRTO. Na presença dessa terceira tentativa é então verificado no *switch*, com base na informação guardada, se o intervalo de chegada é o espectável. No caso afirmativo então o pedido é passado para o *host* protegido. Caso contrário é considerado tráfego malicioso e, portanto, é descartado (*drop*). De salientar que normalmente nos ataques mais simples e automatizados os pacotes enviados a partir do mesmo *host* têm uma cadência fixa pelo que a análise do RTO permite identificar tráfego malicioso que não cumpre os requisitos do protocolo TCP. Antes da ligação ser estabelecida é ainda verificada e atualizada para o caso das mensagens SYN/ACK enviadas pelo *host* protegido, se existe a informação no registo *syn_timestamp_reg*, correspondente ao pedido SYN que deveria ter chegado ao *switch* antes deste SYN/ACK. Quando existe, é então atualizada a informação de que o SYN/ACK já passou pelo *switch*. Similarmente, também na chegada do pacote ACK enviado pelo suplicante é verificado o mesmo registo. Em ambos os casos, se a informação não existir é feito *drop* do pacote. Optou-se por *drop* em vez do envio de uma mensagem de RESET (RST), já que este último computacionalmente implicaria mais esforço por parte do *switch, e* que em termos de segurança poderia dar informação aos atacantes que poderia contribuir para estes identificarem/contornarem as medidas de detenção e mitigação existentes. No caso do *three-way handshake* terminar com sucesso, é monitorizado o RTT máximo registado no *switch* (registo *rtt_reg*) e o RTT máximo por *host-port* (registado em $rtt\_con\_reg_x$). No caso de algum destes RTTs terem sido ultrapassado, é atualizado o respetivo registo com os novos valores máximos. São ainda atualizados os registos $bf_x\_conn\_3hand\_reg$ com o número de vezes que determinada origem efetivou uma conexão com o *host* para o porto de serviço correspondente.

Tabela IX - Índice dos *registers*.

Registos	***Index***
syn_timestamp_reg	hashcrc32(meta.flowID)
rtt_reg	0 para rtt_max e 1 para rtt_medio
rtt_con_regx com x ∈ (1,2)	hashcrc16(boot_timestamp_reg + ((x-1)((2*x)+ (x-1))) , meta.conn$_{(with\ 1\ Ports\ )}$)
pulse_24_reg	-
num_of_new_sources_reg	meta.host_port
Last_timestamp_syn_regx com x ∈ (1,2)	hashcrc16((boot_timestamp_reg + ((2*x)+ (x-1)) , meta.conn$_{(with\ 2\ Ports)}$)
host_3hand_reg	meta.host_port
flow_reg_syn	meta.target_host_index
flow_reg_ack	meta.target_host_index
digest_reg	meta.host_port
boot_timestamp_reg	-
bfx_conn_3handshake_reg com x ∈ (1,2,3)	hashcrc16(boot_timestamp_reg + ((x-1)((2*x)+ (x-1))) , meta.conn$_{(with\ 1\ Ports)}$)

Tabela X – Informação usada nos índices dos r*egisters*.

Metadata	**SYN**	**ACK**	**SYN/ACK**
meta.target_host_index	*index* from table target_host(*ipv4.dstAddr*)		
meta.payload_size	ipv4.totalLen - (((ipv4.ihl) + (tcp.dataOffset)) * 4))		
meta.host_port	*pair_num* from table hostportind(*ipv4.dstAddr*)		
meta.flowID	ipv4.srcAddr ++ ipv4.dstAddr ++ tcp.srcPort ++ tcp.dstPort ++ (tcp.seqNo + meta.payload_size +1)	ipv4.srcAddr ++ ipv4.dstAddr ++ tcp.srcPort ++ tcp.dstPort ++ tcp.seqNo	ipv4.dstAddr ++ ipv4.srcAddr ++ tcp.dstPort ++ tcp.srcPort ++ tcp.ackNo
meta.conn$_{(with\ 1\ Ports)}$	ipv4.srcAddr ++ ipv4.dstAddr ++ tcp.dstPort	ipv4.srcAddr ++ ipv4.dstAddr ++ tcp.dstPort	ipv4.dstAddr ++ ipv4.srcAddr ++ tcp.srcPort
meta.conn$_{(with\ 2\ Ports)}$	ipv4.srcAddr ++ ipv4.dstAddr ++ tcp.srcPort ++ tcp.dstPort	ipv4.srcAddr ++ ipv4.dstAddr ++ tcp.srcPort ++ tcp.dstPort	ipv4.dstAddr ++ ipv4.srcAddr ++ tcp.dstPort ++ tcp.srcPort

Memória

Como anteriormente referido, uma das vantagens da SDN é a possibilidade do uso de dispositivos CPU COTS. No entanto, estes dispositivos têm limitações de memória que têm de ser tidas em conta. Nesse sentido e tendo em conta que a solução aqui apresentada foi desenvolvida com base num dispositivo do tipo CPU COTS, tratando-se de uma máquina virtual cujas características são evidenciadas na Fig. 26, com uma memória disponível de 4096 MB, importa analisar o custo em termos de memória utilizada pela mesma.

Tabela XI - Memória alocada às MAT.

Tabela	Chave	Action	Tamanho	Total (bits)
ipv4_lpm	hdr.ipv4.dstAddr (32bits)	ipv4_forward (57 bits); drop; NoAction	1024	91136
forwarding_arp	hdr.arp.tpa(32bits)	forward_arp (9 bits); drop; NoAction	1024	41984
drop_table		drop		0
blacklist	hdr.ipv4.srcAddr (32bits)	drop_blacklist	1024	32768
port_blacklist	standard_metadata.ingress_port (9bits)	drop_blacklist	SWITCH_PORTS_NUMBER (4)	36
target_host	hdr.ipv4.dstAddr (32bits)	target_host_forward (9 bits)	PROTECTED_TARGET(5)	205
source_host	hdr.ipv4.srcAddr (32bits)	source_host_forward (9 bitsx)	PROTECTED_TARGET(5)	205
hostportind	hdr.ipv4.dstAddr (32bits) standard_metadata.ingress_port (9 bits)	host_portindex (16 bits)	SWITCH_PORTS_NUMBER (4)	228
				166562

Tabela XII - Memória alocada aos *registers*.

Registos	Índice (bit)	Dados (bit)	Tamanho (bit)	Total (bits)
last_timestamp_syn_reg1	32	132	2048	335872
last_timestamp_syn_reg2	32	132	2048	335872
boot_timestamp_reg	32	48	1	80
rtt_reg	32	48	1	80
flow_reg_syn	32	48	PROTECTED_TARGET (5) +1	480
flow_reg_ack	32	48	PROTECTED_TARGET (5) +1	480
syn_timestamp_reg	32	86	REGISTER_SIZE (3072)	264224
bf1_conn_3handshake_reg	32	16	2048	98304
bf2_conn_3handshake_reg	32	16	2048	98304
bf3_conn_3handshake_reg	32	16	2048	98304
rtt_con_reg1	32	128	2048	327680
rtt_con_reg2	32	128	2048	327680
host_3hand_reg	32	192	PROTECTED_TARGET(5) * SWITCH_PORTS_NUMBER(4) .* SWITCH_NUMBER(4) + 1	18144
digest_reg	32	192	PROTECTED_TARGET(5) * SWITCH_PORTS_NUMBER(4) * SWITCH_NUMBER(4) + 1	18144
pulse_24_reg	32	48	1	80
num_of_syn_from_unknown_sources_reg	32	48	PROTECTED_TARGET(5) * SWITCH_PORTS_NUMBER(4) * SWITCH_NUMBER(4) + 1	6480
				1930208

Assim com base nas tabelas XI e XII - onde são apresentadas respetivamente a memória alocada às MAT e aos registos - e na memória alocada aos metadados (1232 bits=1103 bits + 129 bits opcionais), no algoritmo DPSynFloodBlock são alocados 2098002 bits (o que corresponde a 262,25 kB) para armazenamento de informação no dispositivo programável.

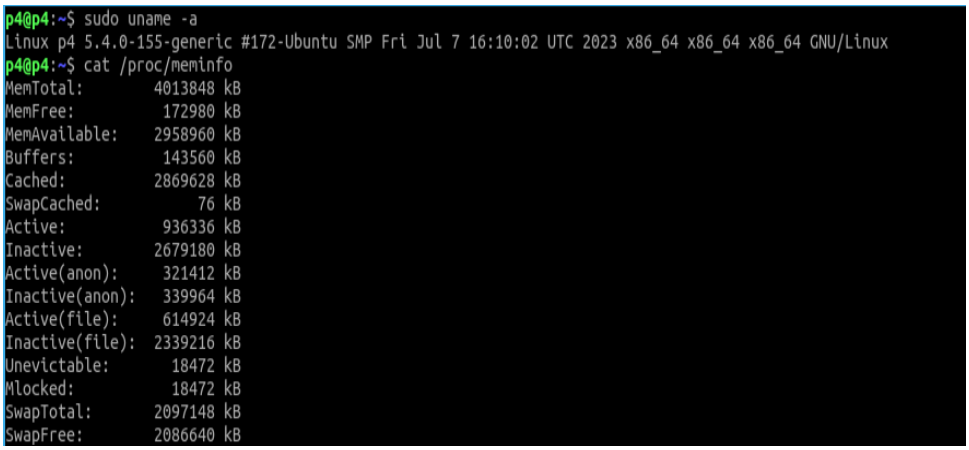


Figura 26 -Informação do sistema do dispositivo programável usado para desenvolvimento e testes do algoritmo DPSynFloodBlock.

4.4 Artifícios implementados para alívio de constrangimentos de P4-16 e SDN

Como descrito no segundo capítulo, apesar das vantagens dos dispositivos de rede programáveis não especializados, também existem desvantagens. Algumas delas estão diretamente relacionadas com as limitações da linguagem usada na programação desses dispositivos. Outra desvantagem, comparativamente aos equipamentos tradicionalmente especializados em funções de rede, é a restrição de capacidade de memória. Assim, foi imprescindível adotar estratégias que assegurassem a realização dos objetivos operacionais do algoritmo, garantindo simultaneamente que a informação necessária estivesse prontamente disponível, mas que esta fosse armazenada de maneira eficiente.

Para alcançar esse propósito, recorreu-se a técnicas de *hashing*, agregação, contagem e filtragem, para armazenar nos registos a informação sobre os dados dos fluxos, garantindo equilíbrio entre a memória consumida e a precisão na identificação dos pacotes. O uso dessas abordagens envolveu a aplicação de estruturas de dados probabilísticas comummente empregadas em contextos de dados contínuos. Mais especificamente, foi usado o algoritmo *count min sketch* [94][95][96] (CMS), para armazenar nos registos bf_x*_conn_3hand_reg* a informação sobre o estabelecimento bem-sucedido de conexões TCP (*three-way handshake* concluído com sucesso). Apesar deste algoritmo ser normalmente usado para determinar a frequência de eventos específicos no contexto de dados e identificar *Heavy Hitters*, o uso deste algoritmo deveu-se à sua simplicidade de implementação e adequação para otimização de memória e processamento de dados em ambientes de redes e fluxos de informações, ainda que não resulte em melhorias no tempo de processamento dos dados.

Comparativamente com o armazenamento da informação em bruto, o uso de algoritmos probabilísticos mantém apenas dados parciais. No entanto, a eficiência e a rapidez nas respostas a consultas justificam a sua adoção. Obviamente, que o armazenamento de dados parciais reflete-se diretamente na precisão das informações mantidas (no caso deste algoritmo, a informação armazenada consiste nas funções *hash* dos dados e à sua frequência). No algoritmo CMS (Fig. 27), para mitigar o desafio das colisões das funções de hash, são empregadas múltiplas funções de hash. Além disso, este algoritmo é caracterizado por não apresentar falsos negativos e possibilita a remoção de entradas ao

decrementar o contador (desde que o seu valor seja superior a zero, o que o torna adequado aos requisitos necessários a este trabalho.

–

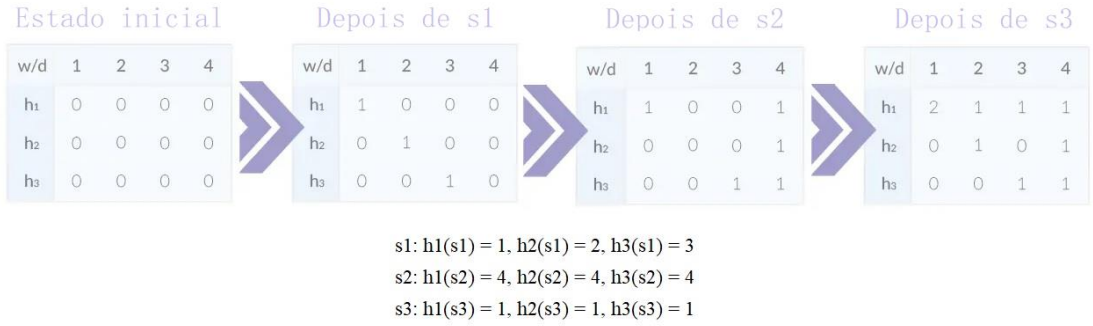


Figura 27 – Exemplo da implementação do algoritmo Count- Min Sketch

Relativamente às técnicas de *hashing* foram usados os algoritmos CRC16 e CRC32 que apesar de serem não criptográficos, são suportados pela arquitetura V1model - utilizada neste trabalho. Os algoritmos de *hashing* criptográficos são normalmente dispendiosos do ponto de vista computacional pelo que é necessário ter em conta este facto, quando se trata do seu uso em implementações que requerem rapidez, como seja o processamento de pacotes em contexto de redes. Apesar de existirem outros algoritmos que minimizam possíveis penalidades relacionadas com o desempenho do processamento de pacotes, como sejam os *Extremely Fast Hash Algorithms* (XXHASH) [97], já usados em alguns trabalhos como seja em [98], o seu uso implicaria o uso de uma arquitetura não padronizada.

Em termos de armazenamento de informação nos registos, visto não ser possível criar uma estrutura de objetos semelhante ao que se pode usar nos metadados - como seja o exemplo da estrutura *digest_t* mostrada na Fig. 24 – foi necessário recorrer a agregação de informação num único campo. Se não se tivesse recorrido a esta opção teria de se desdobrar os registos compostos em outros, contendo somente um tipo de informação, o que significaria o maior consumo do recurso de memória. Na Fig. 28 mostra-se alguns exemplos desta estratégia.

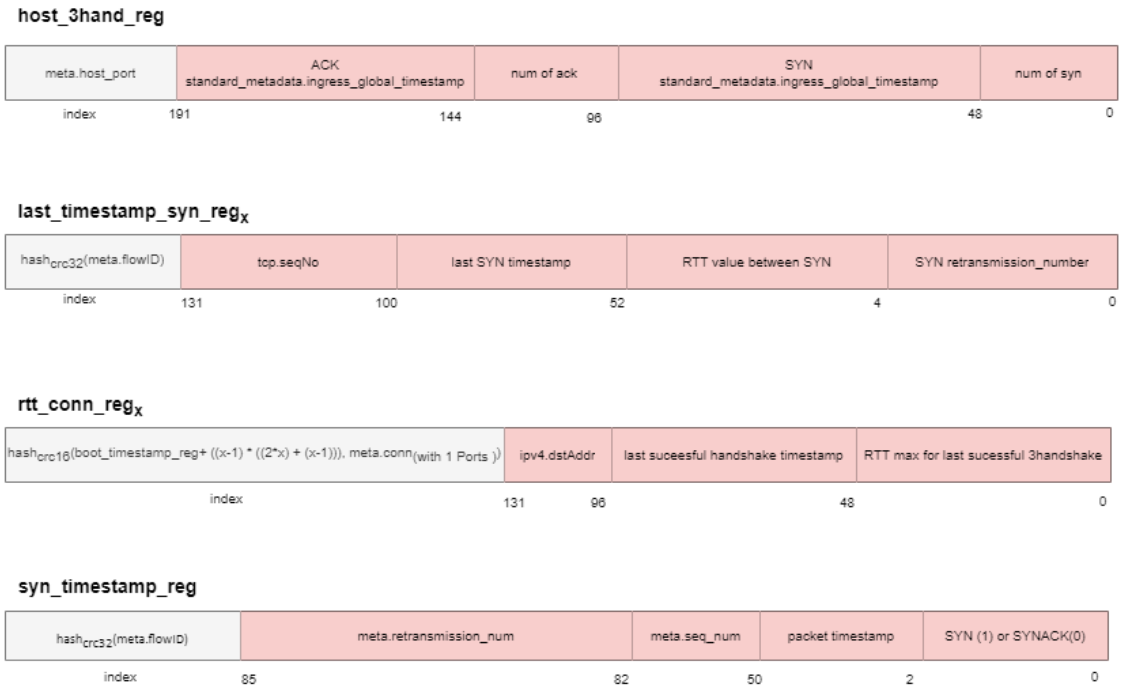


Figura 28 - Registos com agregação de informação.

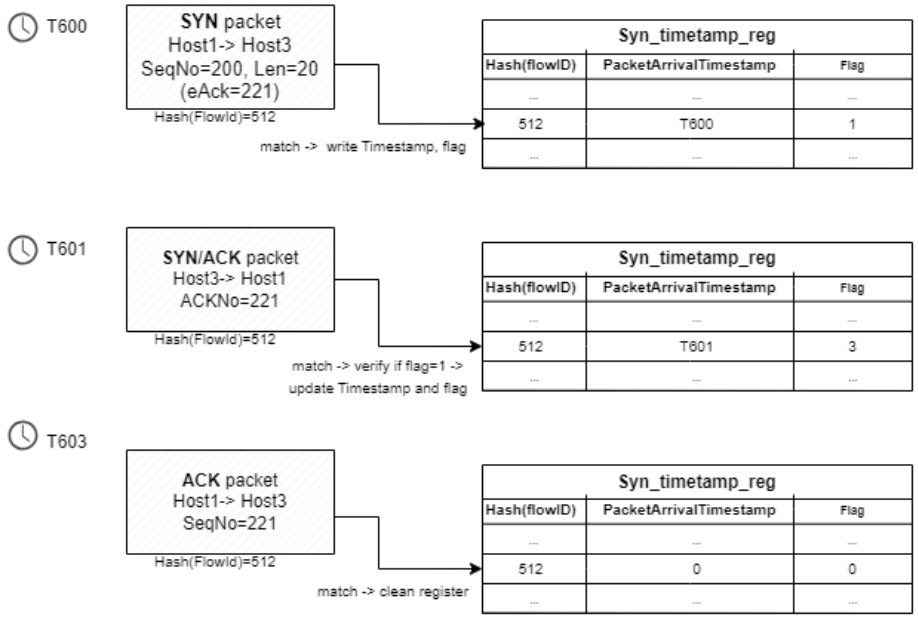


Figura 29- Autolimpeza quando o three-way handshake for bem-sucedido.

Uma outra estratégia implementada, com vista ao uso eficiente deste recurso, foi a autolimpeza dos registos. Nomeadamente, daqueles usados para um fluxo específico, quando a informação neles existente já não tem valor para o algoritmo em questão (por exemplo quando o *three-way handshake* foi bem-sucedido (Fig. 29) ou quando a troca de mensagens para o *three-way handshake* ficou incompleta), bem como para o caso de registos que mantêm informação estatística. Neste último caso é feita a reinicialização das estatísticas baseada no intervalo TIMESTAMP_24.

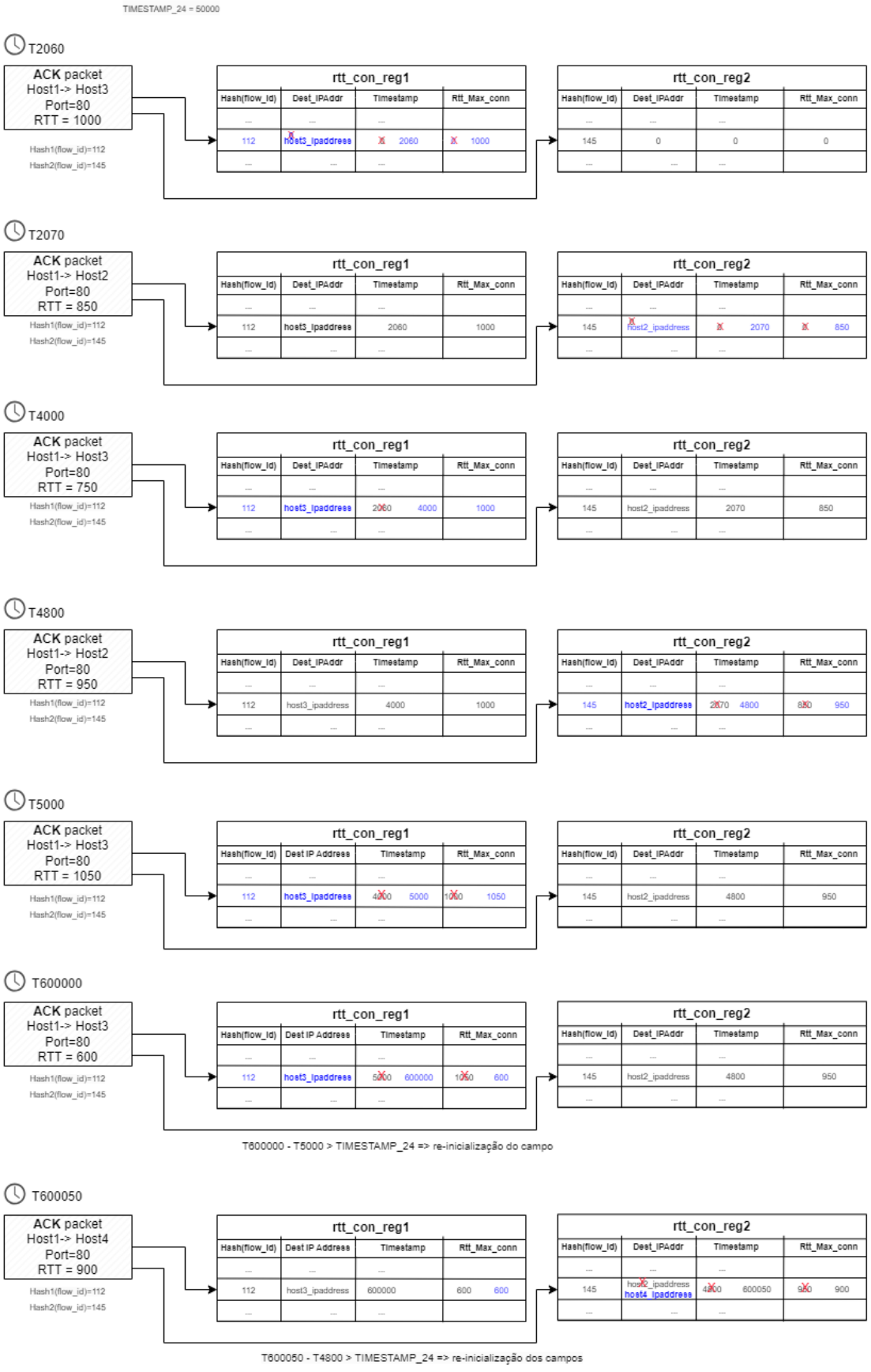


Figura 30 - Uso de tabelas multi-estágio nos registos rtt_con_reg1 e Registos rtt_con_reg2.

No caso de pedidos de ligação incompletos, como seja um ACK que nunca chegou, foi incorporado um mecanismo de expiração dos registos com base no timestamp de chegada do pacote correspondente à última atualização desse registo e um tempo de expiração. Este mecanismo permite que o valor do registo expirado seja substituído por outro que corresponda a esse índice, quando este último está a ser processado. Este artificio, por um lado reduz a carga de trabalho do dispositivo, já que de uma vez só limpa e atualiza, e por outro, ajuda a ultrapassar o constrangimento de acesso da memória do plano de dados pelos dispositivos programáveis, visto que um algoritmo apenas pode aceder à informação de uma MAT, uma única vez, durante o processamento de um pacote. Para esta prova de conceito os tempos de expiração TIMESTAMP_24 e TIMESTAMP_5, são dados por constantes, no entanto, em trabalho futuro terá de ser analisado como determinar os valores ideais adequados ao contexto da rede onde o algoritmo está a ser aplicado, uma vez que uma análise do tráfego a chegar ao dispositivo ao longo do tempo pode ajudar a ajustar estes valores em tempo real de forma a evitar que a memória seja consumida por registos obsoletos.

Como estratégia adicional, com vista a ultrapassar eventuais colisões - apesar de penalizar o recurso de memória - foi adotada o uso de registos duplos, como o caso dos registos *rtt_conn_reg*$_x$ e last_timestamp_syn_reg$_x$. A Fig. 30 mostra como esta estratégia é implementada. No caso de um registo estar ocupado e ser ainda válido (não expirado) é verificado o segundo registo. No caso do segundo registo também estar ocupado e ser ainda válido, é feito o *drop* do pacote. De referir que os registos usam funções *hash* distintas.

Uma outra limitação do P4 que obrigou a uma técnica adicional, é a impossibilidade de atualização de registos dentro de expressões condicionais nas *actions*. Assim por vezes foi necessário recorrer a variáveis para armazenar informação, para que após o uso das expressões condicionais se pudesse atualizar esses registos com o conteúdo dessas variáveis, obrigando assim a utilizar mais expressões comparativamente com outras linguagens. Um exemplo disso é a necessidade do uso da variável *aux_flag* na *action do_update_synack_reg()* devido a não ser possível atualizar o registo *syn_timestamp_reg* dentro das expressões condicionais.

Outra limitação igualmente significativa da linguagem reside na ausência de suporte para divisões diretas por instrução, como observado em outras linguagens. Consequentemente, foi essencial adotar uma estratégia suplementar, considerando esta restrição e a necessidade de estruturar o programa de modo a minimizar o seu impacto. Em situações específicas, foi preciso ajustar o divisor para múltiplos de oito, possibilitando a realização de operações de divisão por meio do deslocamento binário à direita (representado pelo símbolo >>).

Comportamento de DPSynFloodBlock

De acordo com o diagrama de fluxo de DPSyncFloodBlock mostrado no Anexo B, o algoritmo prevê que seja usada a informação proveniente do controlador para bloqueio de eventuais ataques do conhecimento deste, minimizando o eventual impacto no processamento dos pacotes, introduzido pelo tempo de processamento inerente aos pipelines relacionados com a análise de ataque *SYN Flood*, feita unicamente no DP - a partir da informação obtida dos pacotes que chegam ao switch. Assim para qualquer pacote IPv4, após se verificar que não estamos na presença de um ataque conhecido como *land-attack* [99], e antes de qualquer outra análise (com vista à tomada de decisão do DP relativamente a ataques *SYN Flood*), são aplicados os pipelines relativos às tabelas

blacklist e port_blacklist como mostrado na Fig. 31. O pipeline relativo à tabela blacklist descarta todo o tráfego proveniente de determinada origem, já o relativo à tabela port_blacklist descarta todo o tráfego proveniente de um determinado porto do switch.

```
apply {

    if(hdr.ipv4.isValid()){

        if (hdr.ipv4.srcAddr != hdr.ipv4.dstAddr){
            meta.forward_packet=0;

            bit <48> pulse_24;
            boot_timestamp_reg.read(aux,0);

            if (aux == 0){
                aux = standard_metadata.ingress_global_timestamp;
                boot_timestamp_reg.write(0,aux);
                pulse_24_reg.write(0,0);
            }
            pulse_24_reg.read(pulse_24,0);

            bit<48> current_time = standard_metadata.ingress_global_timestamp;
            bit<1> update_24h=0;

            #ifdef DEBUG
                current_time_reg_debug.write(0,standard_metadata.ingress_global_timestamp);
            #endif

            bit<48> value_aux_syn=0;
            bit<48> value_aux_ack=0;
            meta.source_host_hit=0;
            meta.target_host_hit=0;


            if ((!blacklist.apply().hit) && (!port_blacklist.apply().hit)){
                target_host.apply();
                source_host.apply();
                if (((meta.target_host_hit==1)) || ((meta.source_host_hit==1))){
```

Figura 31 - DPSynFloodBlock: Verificações iniciais

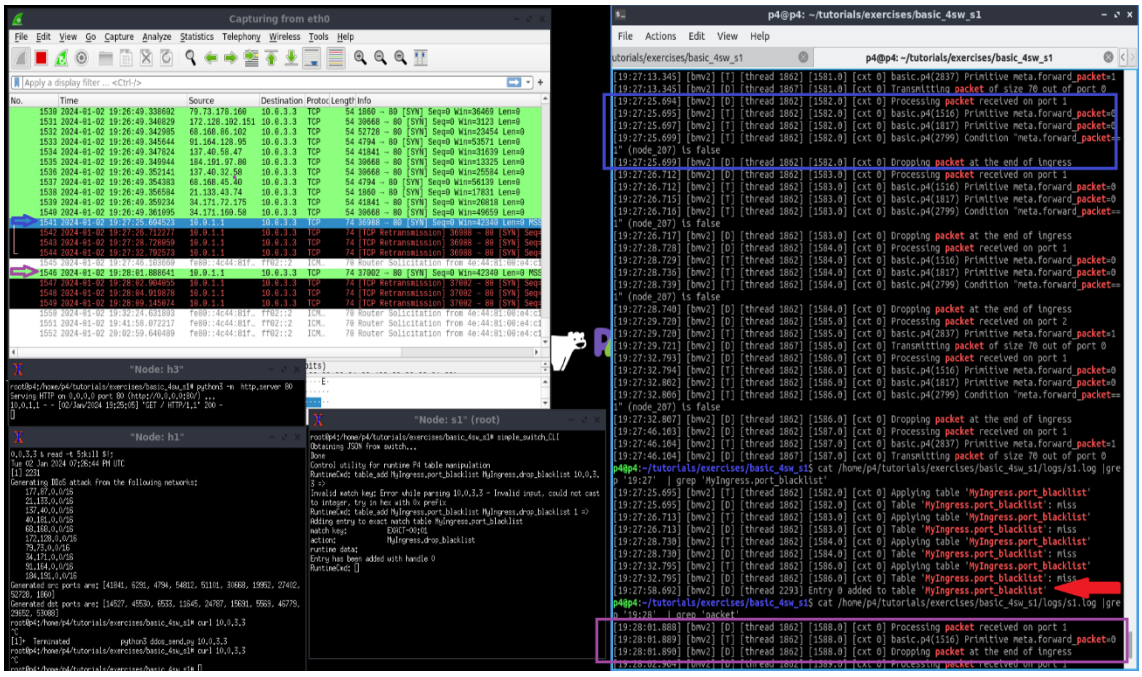


Figura 32 - DPSynFloodBlock: Tempo de processamento de um pedido SYN quando sob ataque.

A Fig. 32 mostra um exemplo do atraso introduzido pelos pipelines de análise do DP para um pedido SYN quando o switch está sob ataque. Nesta, mostra-se que o pacote nº 1541 da captura do *Wireshark* no *host* h1, acontece quando a tabela port_blacklist está vazia e

demora 00:00:00:005 até ser descartado. Enquanto que no caso do pacote nº1546 é descartado em 00:00:00:002, que acontece após a adição à tabela port_blacklist, da linha:

table_add MyIngress.port blacklist MyIngress.drop_blacklist 10.0.3.3

Isto acontece devido ao facto de não ser executado o código correspondente à tomada de decisão do DP relativamente ao ataque de *SYN Flood*.

O comportamento dos pipelines do DPSynFloodBlock responsáveis pela tomada de decisão do DP, varia de acordo com o subtipo de solução, seja ela 0 ou 1, descritas em seguida. Em termos simplificados, é possível afirmar que o atraso no processamento dos pedidos SYN, devido a esses pipelines, representa o ónus associado à tomada de decisão no plano de dados.

Nota: As entradas nas MAT referentes aos testes mencionados na explicação do comportamento da solução 0 e 1, foram definidas de acordo com o Anexo C.

Comportamento sob ataque da solução 0

No subtipo referido como solução 0, pretende estabelecer-se uma defesa para um determinado serviço ou *host*, assemelhando-se a um proxy, com o objetivo exclusivo de proteger o funcionamento desse serviço ou *host*, sem preocupações do ponto de vista da rede no geral. Neste cenário, quando a diferença entre as mensagens SYN e ACK recebidas pelo *switch* atingir o valor predefinido SYN_ACK_THRESHOLD, e a diferença entre essas mensagens no porto do switch atingir o valor predefinido PORT_SYN_ACK THRESHOLD, o tráfego proveniente desse porto, onde o ataque foi identificado, será bloqueado para origens desconhecidas. No entanto, pacotes provenientes de origens conhecidas pelo *switch* continuarão a ser encaminhadas conforme ilustrado no exemplo do Teste 1, descrito nas Fig. 33 - onde é descrito o fluxo de pedidos do teste, sendo de h3 que parte o ataque e h1 o *host* protegido - na Fig. 34 – onde se pode ver os comandos usados no teste - e Fig. 35 - que mostra a captura dos pacotes do teste.

Neste caso, no switch s1 foram configuradas as seguintes entradas para as tabelas target_*host* e source_*host* com a função de proteger h1:

```
MyIngress.target_host: hdr.ipv4.dstAddr=10.0.1.1 => MyIngress.target_host_forward(solution=0, index=1)
MyIngress.source_host: hdr.ipv4.srcAddr=10.0.1.1 => MyIngress.source_host_forward(solution=0, index=1)
```

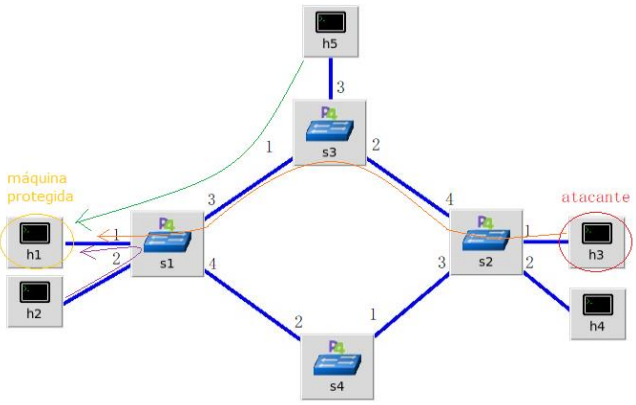


Figura 33 - Descrição dos fluxos de pedidos do Teste 1 aplicável para análise da solução 0.

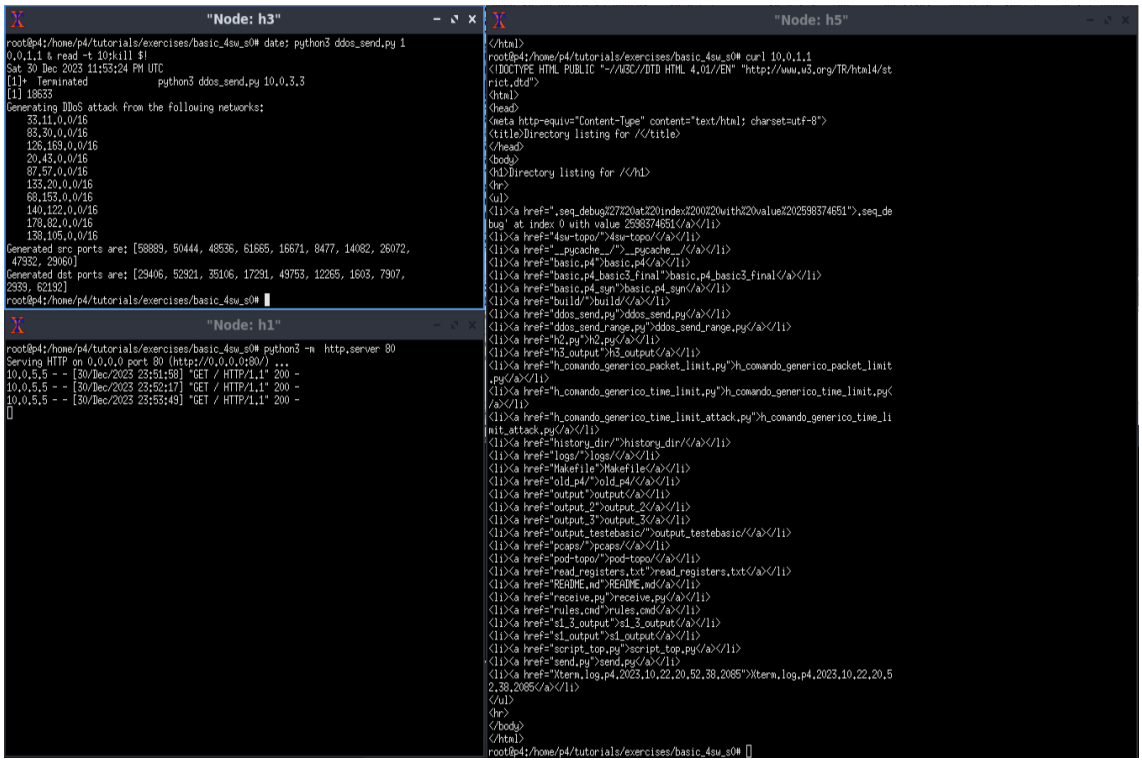


Figura 34 - Teste 1: Saídas do Mininet xterm h1, h3, h5.

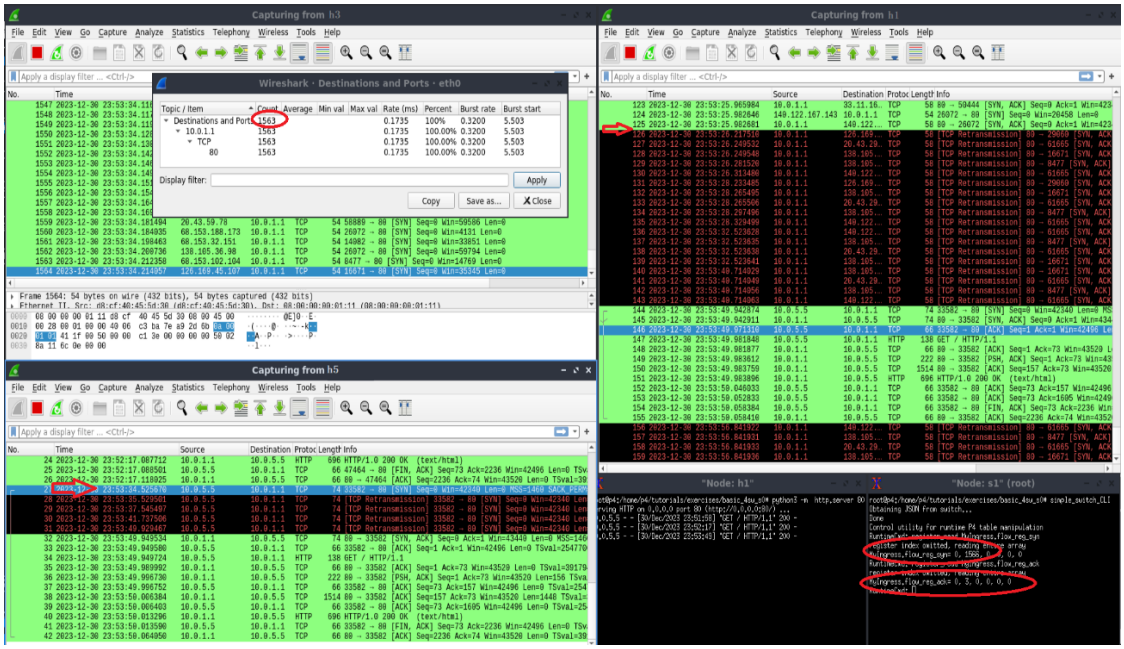


Figura 35 - Captura do *Wireshark* de h1, h3 e h5.

Neste teste, a partir da Fig. 35, é evidente a monitorização (que acontece a partir da análise dos registos *flow_reg_syn* e *flow_reg_ack)*, e consequente deteção e bloqueio, já que apesar do ataque continuar, a partir de h3, até às 23:53:34.214057, h1 recebe o último pedido SYN às 23:53:25:982681. Observamos que, mesmo com um pedido proveniente de h5, feito durante o período de bloqueio, este é encaminhado devido à sua origem

conhecida. Isso ocorre desde que o reenvio desse pedido SYN satisfaça o valor de RTO espectável, apesar da situação de bloqueio no porto. Demonstra-se assim que esta solução, apesar de correr o risco de descartar tráfego legítimo, por descartar pedidos de origem não conhecida, permite o processamento de fluxos originados de fontes previamente conhecidas que chegam ao mesmo porto do switch. Essa abordagem visa evitar sobrecargas tanto no switch quanto no controlador, enquanto assegura a disponibilidade do *host* ou serviço aos seus clientes regulares.

Comportamento sob ataque da solução 1

Para o subtipo referido como solução 1, o objetivo principal é proteger a rede, tendo como foco central uma abordagem semelhante à de uma firewall. Para o Teste 2, que ilustra este subtipo, cujas evidências são mostradas nas Fig. 36 a 43, foram povoadas as seguintes entradas para as tabelas *target_host* e *source_host* no *switch* s1, considerando este como o ponto de entrada da rede que queremos proteger:

```
MyIngress.target_host: hdr.ipv4.dstAddr=10.0.3.3 => MyIngress.target_host_forward(solution=1, index=3)
MyIngress.target_host: hdr.ipv4.dstAddr=10.0.4.4 => MyIngress.target_host_forward(solution=1, index=4)
MyIngress.target_host: hdr.ipv4.dstAddr=10.0.5.5 => MyIngress.target_host_forward(solution=1, index=5)
MyIngress.source_host: hdr.ipv4.srcAddr=10.0.3.3 => MyIngress.source_host_forward(solution=1, index=3)
MyIngress.source_host: hdr.ipv4.srcAddr=10.0.4.4 => MyIngress.source_host_forward(solution=1, index=4)
MyIngress.source_host: hdr.ipv4.srcAddr=10.0.5.5 => MyIngress.source_host_forward(solution=1, index=5)
```

Neste caso e contrariamente à solução 0, verifica-se que apesar de *host* h3 já ter efetuado uma conexão TCP com sucesso antes do ataque proveniente de h3 - que chegou no mesmo porto do switch onde o ataque está a chegar - os pedidos de SYN são bloqueados no porto correspondente, durante o tempo em que o switch considerar que está sob ataque. Na prática o algoritmo bloqueia todo o tráfego proveniente de determinado porto, salvaguardando a continuação do funcionamento da rede, como se pode verificar nas evidências do Teste 2 mostrado na Fig. 38. Nela observa-se que o pedido efetuado por h1 ao serviço *http* de h3 através do comando *"curl 10.0.3.3"* correspondente ao pacote nº 1293 da captura do *Wireshark* de h1, não é encaminhado, sendo descartado devido ao porto 1 do *switch* s1 estar bloqueado. De salientar, que no mesmo *switch* o pedido feito pelo *host* h2 que chega no porto 2 do *switch* s1 é encaminhado sem problemas para h3, como podemos ver na captura de *Wireshark* para h3, da mesma figura.

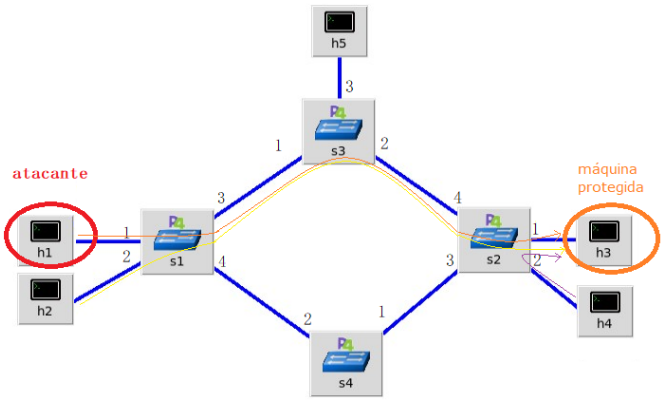


Figura 36 - Descrição dos fluxos do teste 2 aplicável para analise da solução 1.

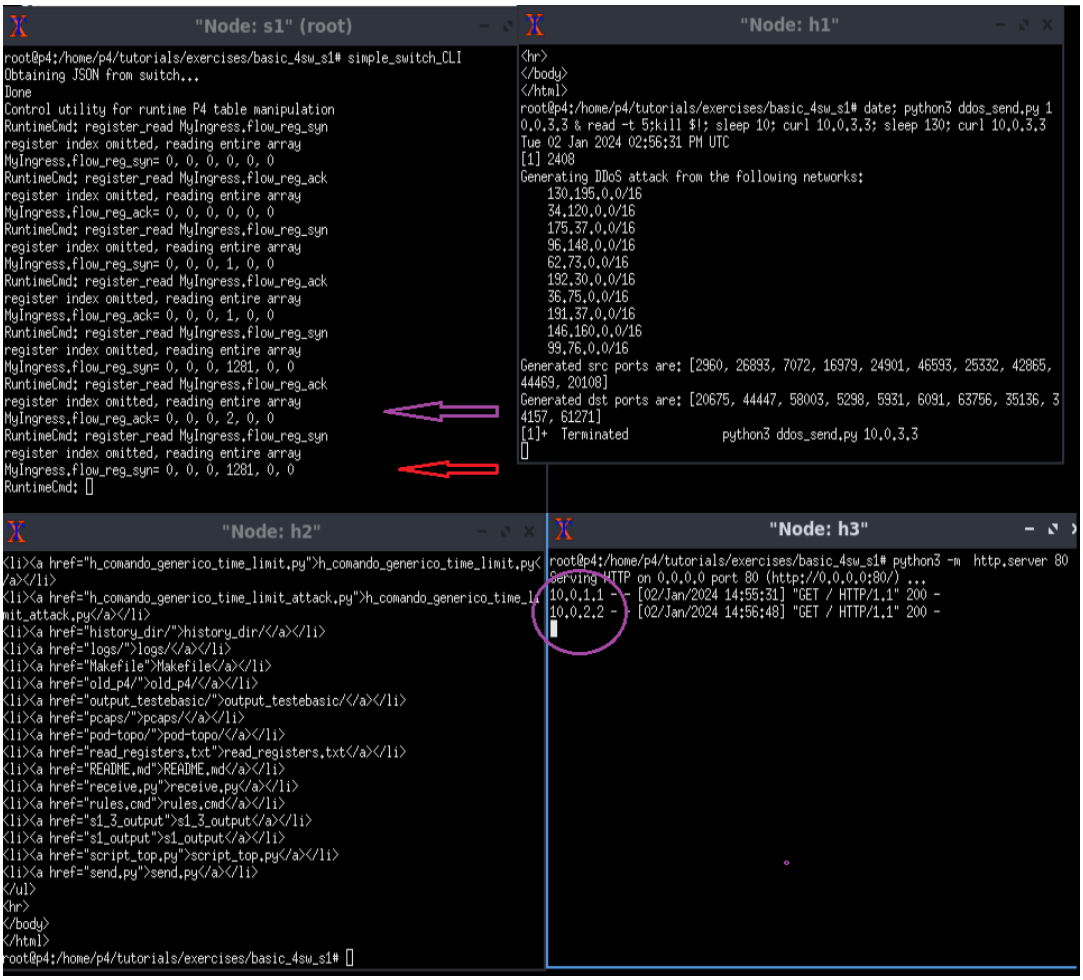


Figura 37 - Teste 2 (durante ataque): Saídas do Mininet xterm h1, h3, h5, s1.

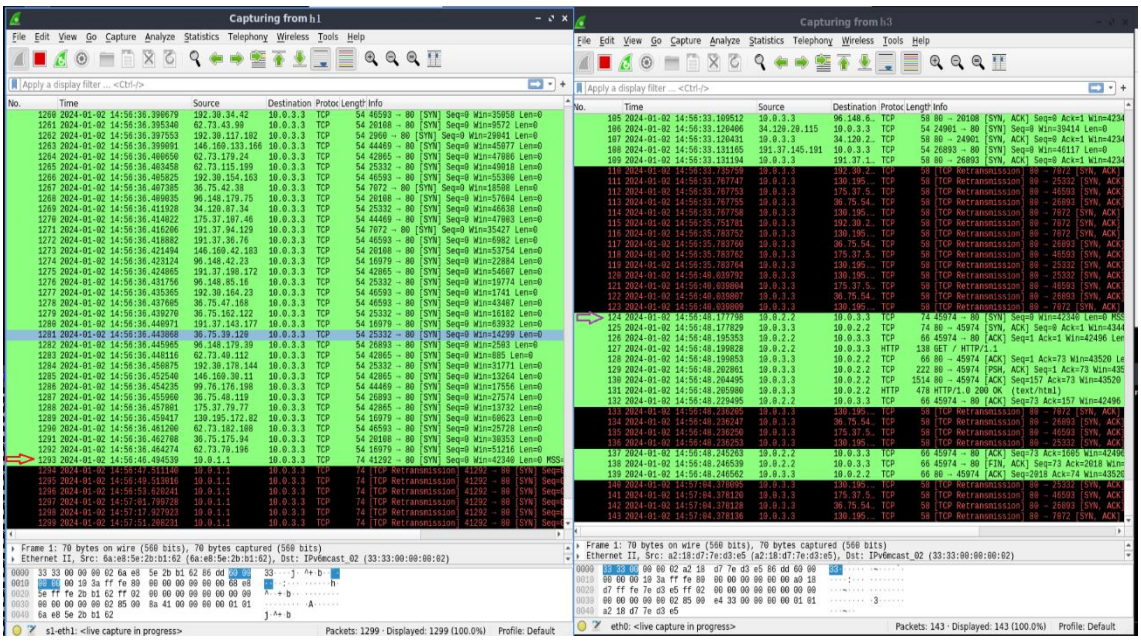


Figura 38 - Teste 2 (durante ataque): Captura do *Wireshark* em h1 e h3 durante o ataque de *SYN Flood*.

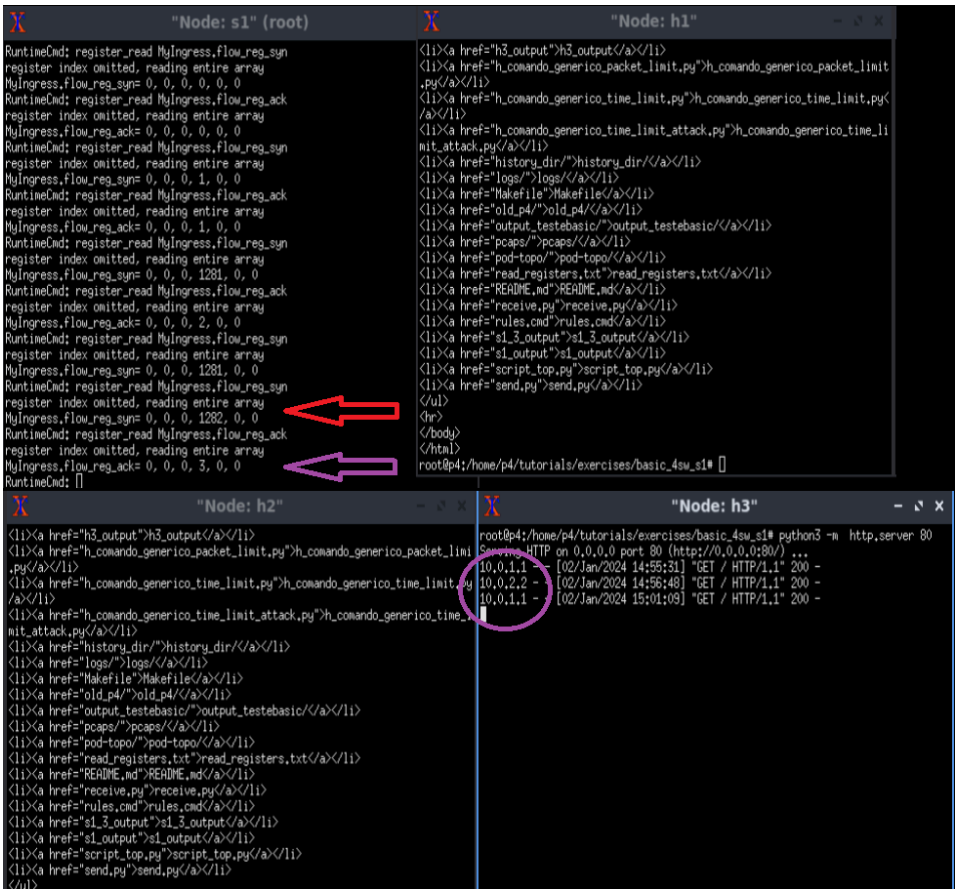


Figura 39 - Teste 2 (decorrido intervalo TIMESTAMP_5 após ataque): Saídas do Mininet xterm s1, h1, h2, h3 após pedido de h1.

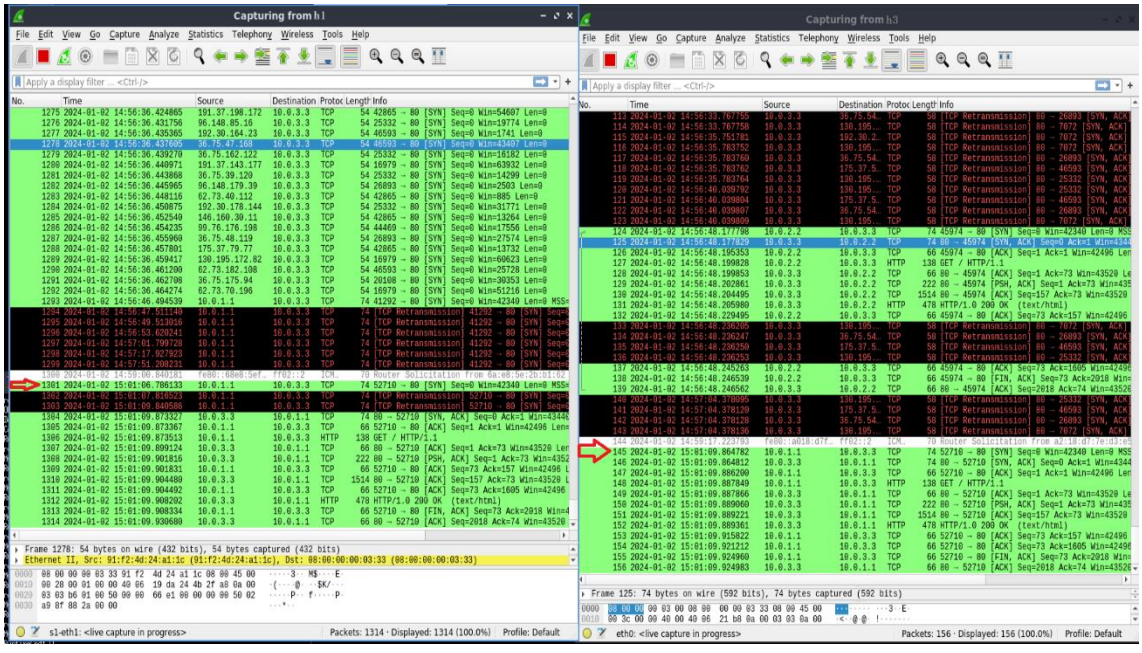


Figura 40 - Teste 2 (decorrido intervalo TIMESTAMP_5 após ataque): Captura do *Wireshark* de h1 e h3 após pedido de h1.

Decorrido o intervalo de tempo TIMESTAMP_5 após o ataque não se verificar, inicia-se a fase de prevenção, na qual os pacotes SYN são direcionados para os pedidos que atendam ao comportamento descrito na Fig. 25. O primeiro pedido SYN, bem como a seu primeiro reenvio são descartados e seu primeiro reenvio são descartados, sendo encaminhados apenas os reenvios subsequentes que cumpram o valor esperado de RTO, conforme ilustrado na Fig. 40.

Nesta solução o comportamento de prevenção é apenas aplicável aos pedidos provenientes do porto envolvido no ataque, como se pode verificar no exemplo ilustrado na Fig. 41 e 42. Nele, verifica-se que os pedidos correspondente aos pacote nº 3 e 16 da captura de *Wireshark* de h2, são encaminhados sem qualquer restrição, contrariamente ao verificado na solução 0, onde após TIMESTAMP_5 todos os pedidos efetuados ao *host*, independentemente do porto de chegada, são afetados pela verificação do módulo de prevenção com o comportamento descrito na Fig. 25.

Tal como já descrito anteriormente, decorrido o intervalo de tempo TIMESTAMP_24 do arranque inicial do switch, as condições de ataque são limpas de forma a evitar possíveis constrangimentos do funcionamento deste. Este comportamento pode ser comprovado no comportamento ilustrado nos exemplos das Fig. 41 a 45, onde se verifica que após decorrido um intervalo de tempo, múltiplo de TIMESTAMP_24, após o arranque do switch, os registos relacionados com informação de ataque são limpos, e todo o processo de monitorização é reiniciado.

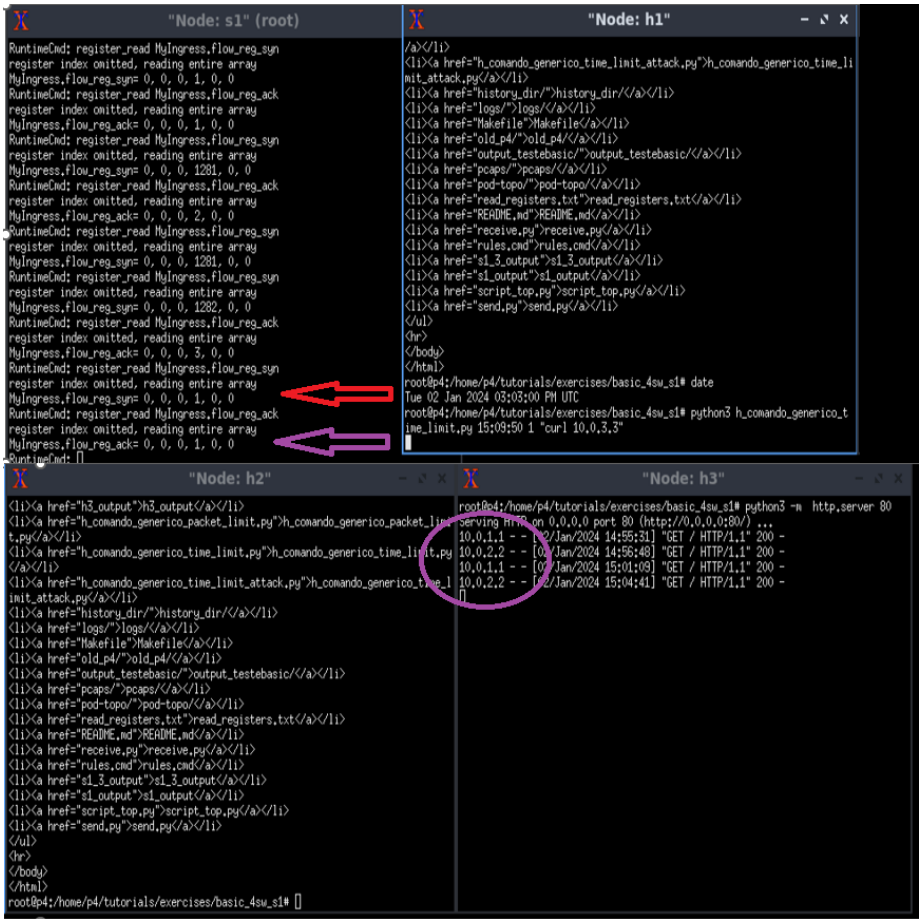


Figura 41 - Teste 2 (decorrido intervalo TIMESTAMP_24 do arranque inicial do switch): Saídas do Mininet xterm s1, h1, h2, h3 após pedidos de h1 e h2.

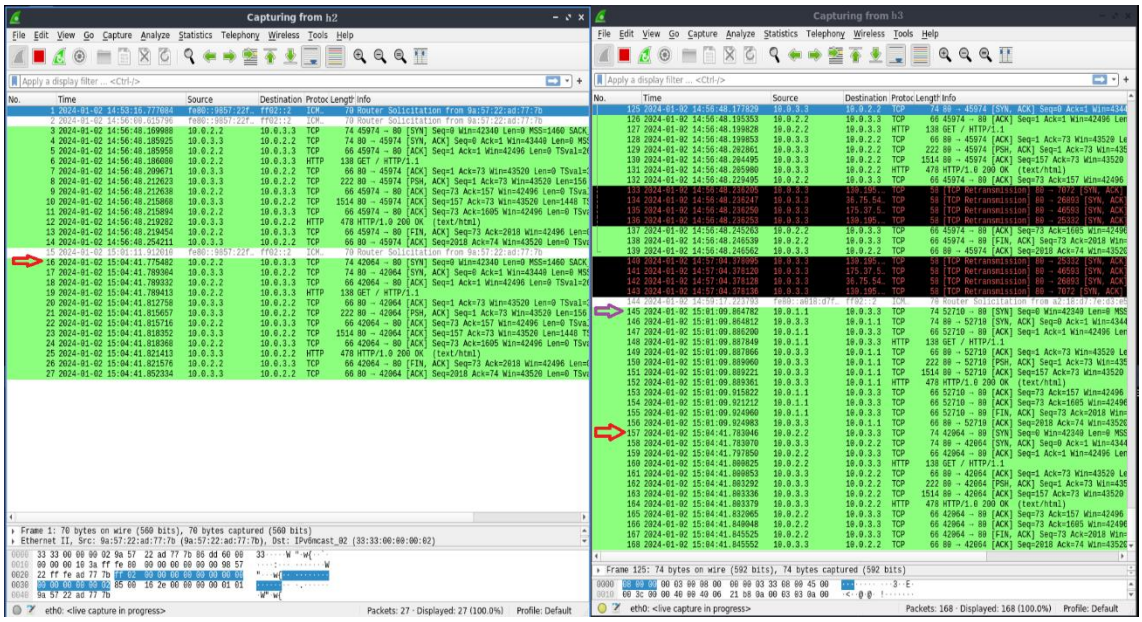


Figura 42 - Teste 2 (decorrido intervalo TIMESTAMP_24 do arranque inicial do switch): Captura do *Wireshark* de h2 e h3 após pedidos de h1 e de h2.

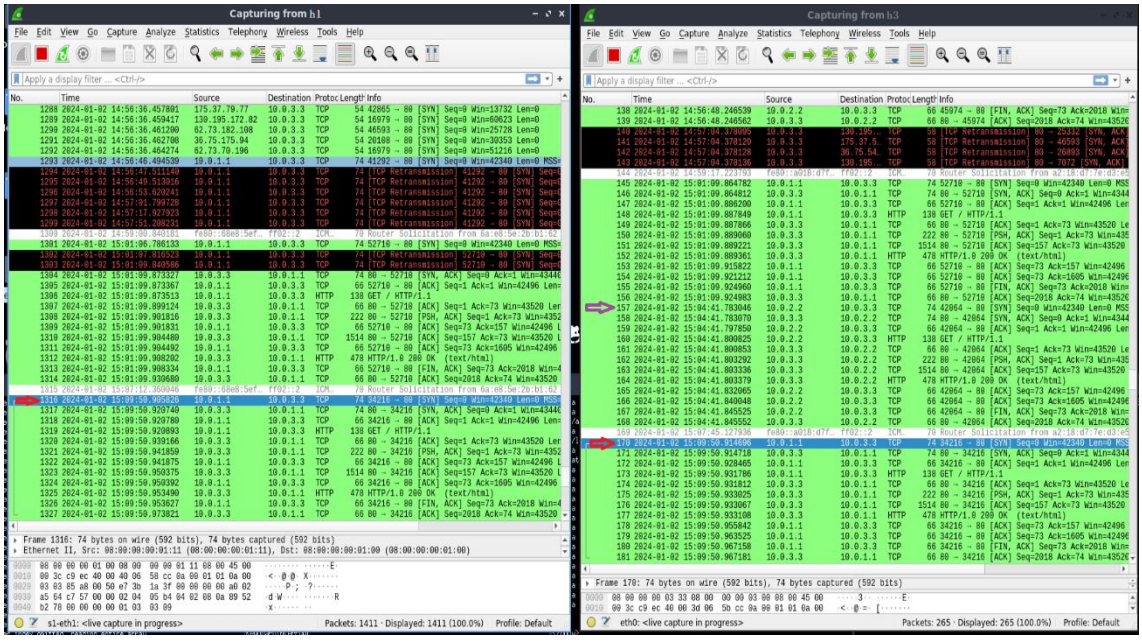


Figura 43 - Teste 2 (decorrido intervalo TIMESTAMP_24 do arranque inicial do switch): Captura do *Wireshark* de h1 e h3, após pedidos de h1 e de h2.

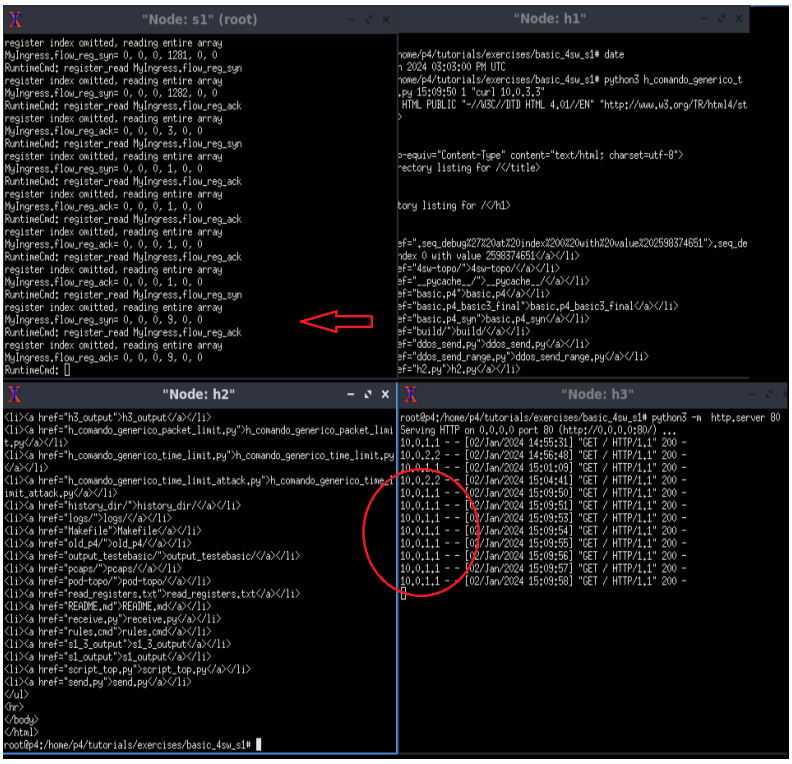


Figura 44 - Teste 2 (decorrido intervalo TIMESTAMP_24 do arranque inicial do switch): Saídas do Mininet xterm s1, h1, h2, h3 após pedidos de h1 de h2.

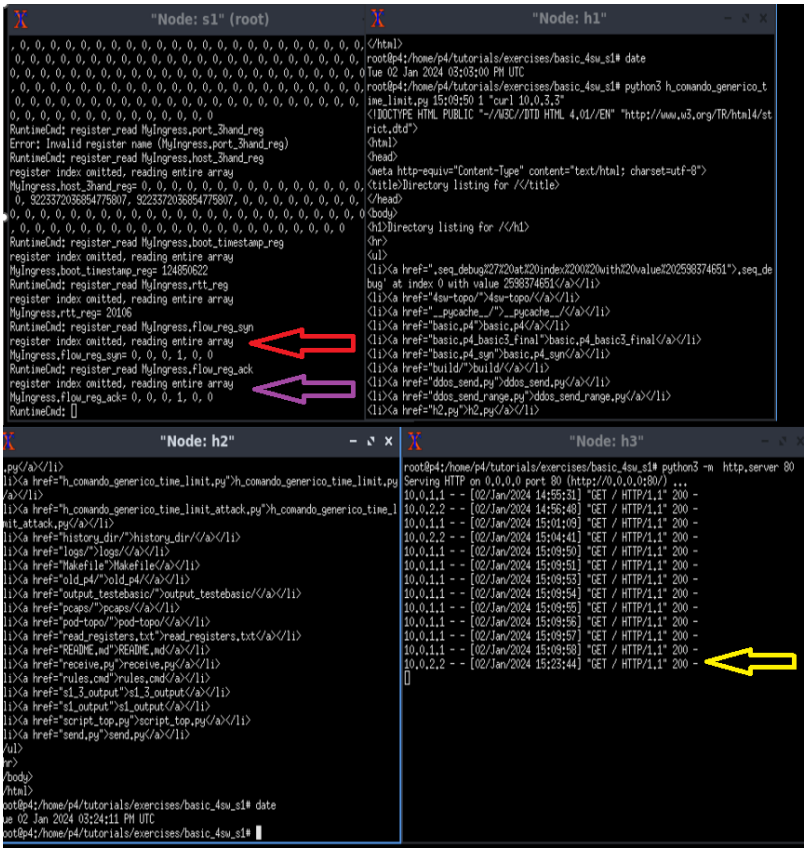


Figura 45 - Teste 2 (decorrido intervalo 2 * TIMESTAMP_24 do arranque inicial do switch): Saídas do Mininet xterm s1, h1, h2, h3.

Similarmente ao subtipo 0, também neste subtipo se corre o risco de descartar tráfego legítimo. No entanto é o custo para garantir o bom funcionamento não só do *host*/serviço que se quer proteger, mas também da rede interna. Neste caso o bloqueio dos pedidos provenientes de um determinado porto permite que nem o *switch* nem o controlador sejam assoberbados com o processamento destes pedidos, e que pedidos provenientes de outros portos do switch possam ser processados.

Capítulo 5
Avaliação

Um dos objetivos deste trabalho era a construção de um algoritmo simples do ponto de vista de implementação. Por essa razão decidiu-se compará-lo a um SYN Proxy, nomeadamente ao algoritmo disponível em [100]. Similarmente ao algoritmo DPSynFloodBlock, também este SYN Proxy recorre a um algoritmo probabilístico, neste caso usa um filtro Bloom, para a sua implementação. Este SYN Proxy implementa um mecanismo de mitigação no plano de dados para ataques de *SYN Flood*, usando autenticação SYN com redefinição na primeira conexão. Após o primeiro pedido SYN para se conectar ao *host*/serviço, o *switch* responde com um RST - para que seja gerada nova tentativa de conexão, de acordo com a implementação do protocolo TCP - e, em seguida, adiciona o endereço IP da origem do pedido ao filtro Bloom, criando uma lista de origens permitidas (*whitelist).* Uma origem legítima tentará reconectar-se, e aquando da chegada do reenvio do pedido, o filtro Bloom será verificado. Caso seja confirmada a existência de tentativa prévia correspondente à entrada do pedido inicial no filtro, a conexão é encaminhada para o *host*/serviço.

Essa implementação, onde é estabelecida uma autenticação SYN, é considerada uma abordagem de mitigação exclusivamente do plano de dados, já que é implementada uma *whitelist* através de um registo, cuja concretização é feita através do filtro Bloom sem recorrer ao controlador. Por isso serve o propósito de comparação, já que similarmente se pretende que DPSynFloodBlock seja uma abordagem que não dependa do controlador.

Assim para esta avaliação foi considerado a topologia mostrada na Fig. 46, construída no Mininet com um switch BMv2:

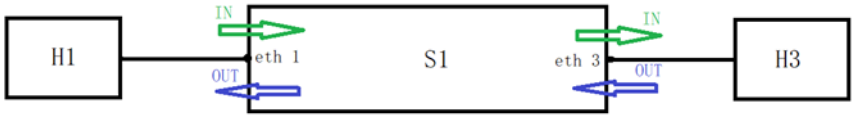


Figura 46 - Topologia considerada na comparação de DPSynFloodBlock e SYN Proxy com filtro Bloom.

Para a avaliação foram efetuados três testes para cada algoritmo na qual foram empreendidos os comandos especificados no Anexo D.

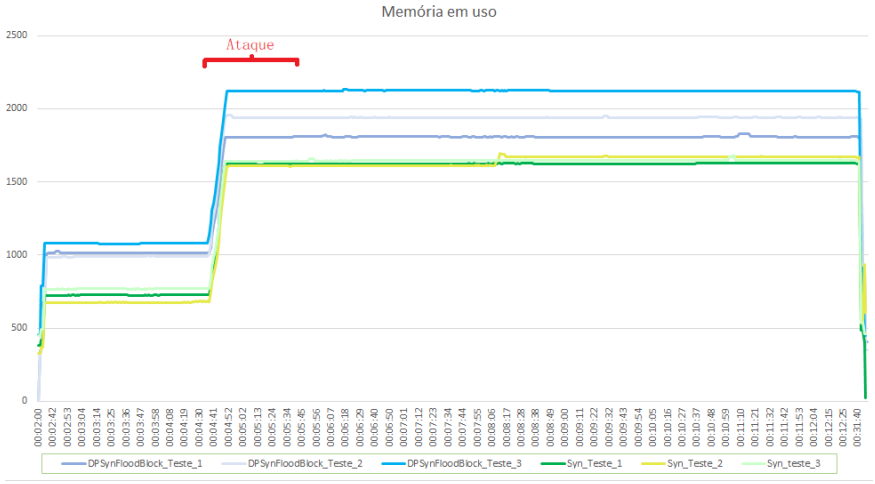


Figura 47 - Gráfico de memória em uso obtida nos testes com os algoritmos DPSynFloodBlock e SYN Proxy.

Através dos resultados obtidos a partir dos valores de memória em uso, recolhidos através da ferramenta *top*, obteve-se o gráfico mostrado na Fig. 47. Este gráfico evidencia que o algoritmo DPSynFloodBlock requer mais memória para a sua execução comparativamente ao SYN Proxy.

Tabela XIII - Número de pacotes nas interfaces dos switches.

Algoritmo	Teste #	Ataque	TCP Flag	h1	s1_eth1_in	s1_eth1_out	s1_eth3_in	s1_eth3_out	h3
DPSynFloodBlock	1	11:34:40 - 11:34:45	syn	316053	1437	0	0	50	50
			syn + ack	0	0	0	70	0	70
			ack	0	0	0	0	0	0
			rst	0	0	0	0	0	0
			retrans	0	0	0	22	0	22
	2	13:02:01 - 13:02:06	syn	263248	1305	0	0	49	49
			syn + ack	0	0	0	70	0	70
			ack	0	0	0	0	0	0
			rst	0	0	0	0	0	0
			retrans	0	0	0	22	0	22
	3	14:12:10 - 14:12:15	syn	279606	1424	0	0	49	49
			syn + ack	0	0	0	70	0	66
			ack	0	0	0	0	0	0
			rst	0	0	0	0	0	0
			retrans	0	0	0	22	0	22
syn_proxy	4	19:12:41 - 19:12:46	syn	432468	3409	0	0	1207	1207
			syn + ack	0	0	0	1139	0	1139
			ack	0	0	0	0	0	0
			rst	0	0	0	0	360	360
			retrans	0	0	0	22	0	22
	5	19:52:41 - 19:52:46	syn	299634	3165	0	0	1032	1032
			syn + ack	0	0	0	981	0	981
			ack	0	0	0	0	0	0
			rst	0	0	0	0	345	345
			retrans	0	0	0	22	0	22
	6	21:22:41 - 21:22:46	syn	312467	3087	0	0	980	980
			syn + ack	0	0	0	930	0	930
			ack	0	0	0	0	0	0
			rst	0	0	0	0	329	329
			retrans	0	0	0	22	0	22

A partir das capturas de *Wireshark* para cada teste, conclui-se que relativamente ao número de pacotes, mostrado na Tabela XIII, que chegam ao *host*/serviço, o algoritmo DPSynFloodBlock é mais eficiente a bloquear os pacotes maliciosos, já que só passam os pedidos SYN até ao valor de SYN_ACK_THRESHOLD, enquanto com o SYN Proxy passam todos os pedidos cujo endereço IP de origem e o porto de destino já tenham sido registados no bloom filter - mesmo que o seu número de sequência do pedido seja distinto.

Foi também analisada a performance dos dois algoritmos, a partir de testes com a ferramenta *iperf* e com o *ping*.

O gráfico da Fig. 48 mostra, para os dois algoritmos, os resultados dos testes de iperf, onde se executou os seguintes comandos:

h3# iperf -s

h1# iperf -c 10.0.3.3 -t 60s

A partir deste gráfico, verifica-se que o SYN Proxy (ilustrado a azul) tem resultados de *Bandwidth* maiores que o DPSynFloodBlock (ilustrado a verde). Este resultado está de acordo com o esperado devido ao número de instruções do código do SYN Proxy, bem como o número de pipelines a executar serem bastante inferiores ao número usado pelo DPSynFloodBlock. Além disso, a informação recolhida e analisada para a tomada de decisão é também bastante inferior.

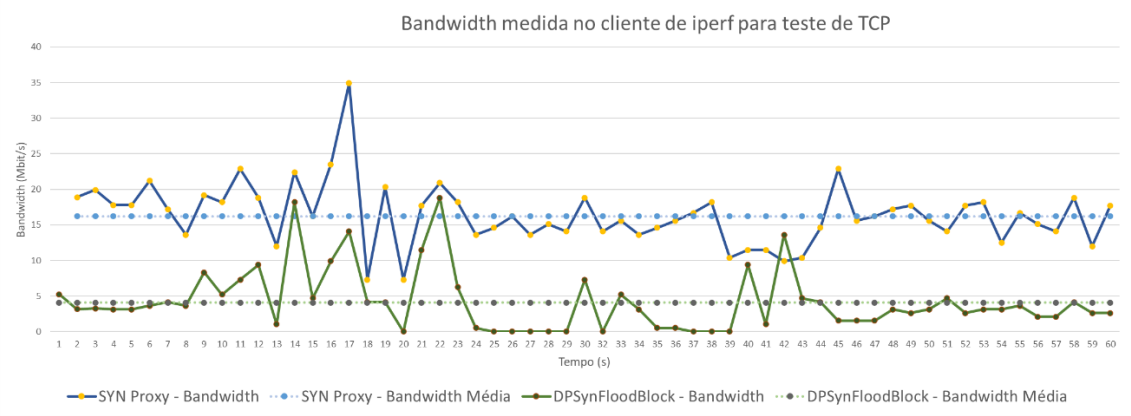


Figura 48 - Teste com Iperf para os algoritmos DPSynFloodBlock e SYN Proxy.

Também, a partir do gráfico da Fig. 49, correspondente aos resultados dos testes de *ping*, verifica-se que a o valor de RTT para a DPSynFloodBlock (ilustrado a verde) é superior aos valores do outro algoritmo (ilustrado a azul), indicando-nos que para o primeiro existe uma latência maior no processamento dos pacotes.

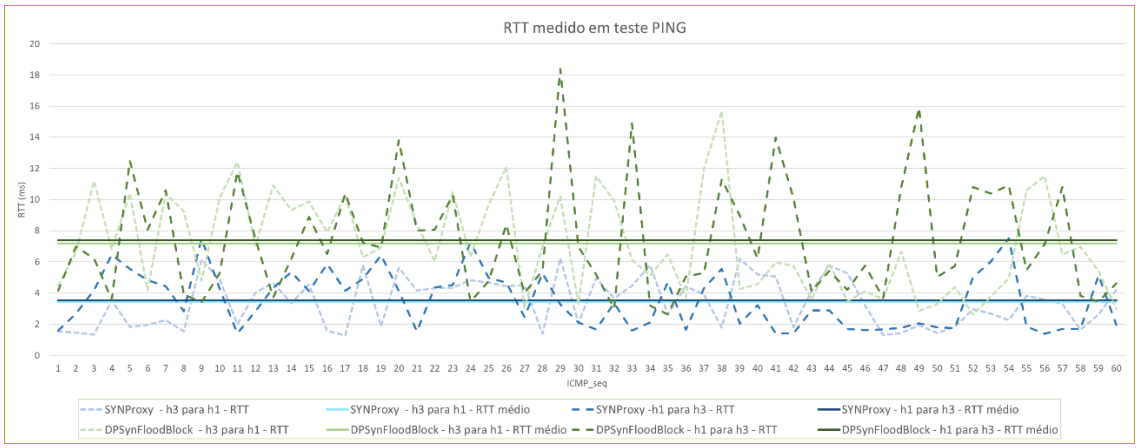


Figura 49 - Teste com *ping* para os algoritmos DPSynFloodBlock e SYN Proxy.

O algoritmo SYN Proxy comparativamente ao DPSynFloodBlock, tem menos pipelines MAT, pelo que o resultado anterior é o esperado, visto que o número de processos pelos quais o pacote tem de passar é inferior ao algoritmo desenvolvido neste trabalho. No entanto, este resultado levanta a questão se esta latência é só influenciada pelo número de pipelines que o algoritmo tem, ou se o número de instruções por si só terá uma influência de peso.

Para confirmar que este facto tem um papel fundamental na latência, nomeadamente no tempo médio de RTT de uma conexão TCP, executou-se o mesmo teste de *ping* para o algoritmo DPSynFloodBlock mas eliminando todas as instruções usadas exclusivamente para o auxílio de depuração - comummente designado como *debug*- mantendo assim o número de pipelines, mas diminuindo o número de instruções. Observando os resultados deste teste, mostrado a laranja no gráfico da Fig. 50, com os resultados do teste anterior para o mesmo algoritmo com instruções de *debug*, mostrado a azul no mesmo gráfico, verifica-se que o número de instruções tem influência na latência.

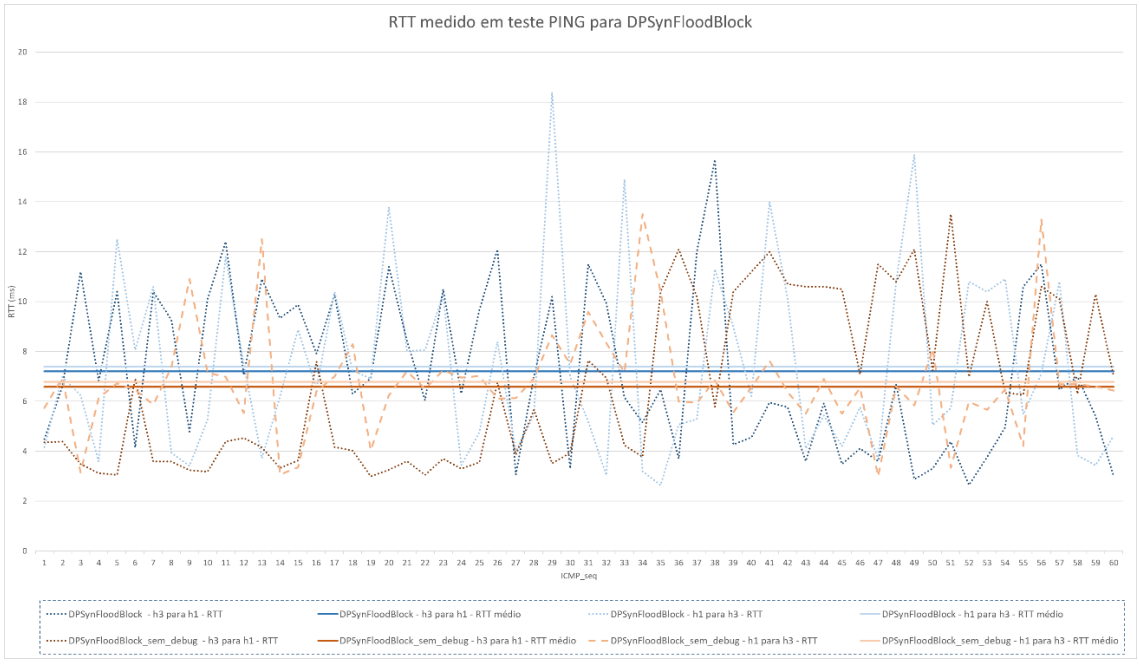


Figura 50 - Teste com *ping* para o algoritmo DPSynFloodBlock com e sem instruções de *debug*.

Assim, através da experimentação com o código P4 dos dois algoritmos, DPSynFloodBlock e SYN Proxy, foi observado que a necessidade de coletar e analisar uma maior quantidade de informações por meio do algoritmo resulta na exigência de um código mais extenso e complexo para a sua implementação. Esta circunstância, por conseguinte, sugere a possibilidade de uma degradação de desempenho na rede, decorrente da latência introduzida pela execução do referido algoritmo, cuja hipótese os resultados anteriores corroboram. Consequentemente, para além da utilização de memória, conclui-se que também a latência, constitui um ónus necessário para obter um conjunto de dados através da programação, capaz de permitir a tomada de decisão dos dispositivos programáveis, visando à mitigação de possíveis ataques. Torna-se, portanto, manifesta a importância da análise dos recursos disponíveis e consumidos pela programação do plano de dados, assim como os encargos relacionados com a latência que uma programação mais complexa acarreta.

No contexto de rede SDN real, é imperativo investigar tais variáveis e considerar as várias alternativas disponíveis para a programação dos dispositivos programáveis para alcançar o objetivo de mitigar ataques DDoS, não descartando que outras soluções aplicáveis no plano de controlo e no plano de aplicação, em colaboração com as soluções do plano de dados, poderão contribuir significativamente para uma programação mais eficiente dos dispositivos programáveis.

Não obstante, este estudo, demonstrou a viabilidade de tornar os dispositivos programáveis mais inteligentes através da linguagem P4, capacitando-os a monitorar, detetar, mitigar e prevenir ataques DDoS, utilizando somente informações extraídas a partir dos pacotes de dados e informação mínima inicial fornecida pelo controlador.

A solução aqui apresentada, DPSynFloodBlock, contribui de maneira significativa para assegurar não apenas a disponibilidade do *host* em questão ou da rede à qual pertence, mas também do próprio sistema de SDN. Isso ocorre porque, ao realizar a tomada de decisões no plano de dados, evita-se a sobrecarga do controlador com solicitações de pacotes *Packet_in*. Essas solicitações poderiam consumir considerável largura de banda na interface SBI, além de recursos críticos do controlador, como CPU e memória. Desta forma podemos afirmar que DPSynFloodBlock cumpre com o objetivo de melhorar a segurança, contribuindo para a redução do risco decorrente da ameaça de interrupção de serviço no sistema SDN.

Capítulo 6
Conclusão

6.1 Sumário

A ideia central da arquitetura SDN é a separação do plano de controlo do plano de dados. Ao separar estes planos, a SDN traz maior programabilidade à rede. Esta característica leva a uma melhoria do controlo do tráfego e uma maior facilidade na configuração de dispositivos de rede programáveis. Como consequência, esta flexibilidade abre espaço para a inovação na conceção, na operação e na segurança da rede. O estudo aqui apresentado partiu desta ideia, focando-se nas questões de segurança desta arquitetura.

Neste trabalho foi analisado detalhadamente o conceito de redes definidas por *software* (SDN). Particularmente foi feita uma descrição da arquitetura SDN e funcionamento de cada plano e abordadas as particularidades da segurança desta arquitetura.

Adicionalmente e de forma mais detalhada foram analisados os ataques DDoS em SDN, sobre os quais este estudo se centra, e respetivos mecanismos de defesa usados em SDN para os mitigar.

Para um melhor enquadramento do estudo foram revistos os conceitos associados à linguagem P4 e analisadas as vantagens do plano de dados baseado em P4.

Posteriormente foram apresentados os objetivos deste estudo, bem como a metodologia utilizada, detalhando a experimentação inicial efetuada, usada para melhor compreensão dos conceitos subjacentes a este estudo.

Foi apresentada a solução DPSynFloodBlock, desenvolvida utilizando a linguagem P4, destinada à monotorização, deteção, mitigação e prevenção de ataques DDoS *SYN Flood*, cuja simplicidade de implementação os torna uma ameaça preocupante no contexto de SDN. Nesta solução recorreu-se a técnicas de análise estatística, usadas para detetar anomalias e para armazenar informação. Além disso, empregou-se ainda estratégias de gestão de largura de banda, limitando o número de solicitações ao controlador, mas ainda assim possibilitando a alteração das MATs de forma a mitigar as anomalias. No entanto, mesmo sem intervenção direta do controlador, essa solução capacita os dispositivos programáveis a conter tais ataques específicos, desde que dados iniciais pertinentes à rede e aos objetivos do algoritmo estejam presentes nas MATs. A natureza desse conteúdo inicial nas MATs oferece a possibilidade de selecionar a defesa de um *host* ou serviço específico ou de bloquear o ataque o mais próximo possível da sua origem, contribuindo, assim, para a otimização do funcionamento interno da rede SDN.

Foi ainda efetuada uma avaliação da solução onde se verificou que por um lado a quantidade de informação necessária para alcançar os objetivos tem um peso significativo na memória usada, e por outro que o aumento do número de instruções da programação tem impacto negativo em termos de performance de rede.

A vantagem distintiva do DPSynFloodBlock relativamente a outras soluções implementadas no plano de dados e baseadas nesse mesmo plano reside na sua abordagem de aplicar a mitigação exclusivamente aos fluxos que chegam ao dispositivo programável após a deteção do ataque, alcançada por meio da monitorização dos fluxos que transitam

nesse mesmo dispositivo. Além disso, integra mecanismos de prevenção pós-ataque para assegurar a continuidade das operações normais do dispositivo. Também oferece a capacidade de programação do dispositivo para proteger uma rede interna ou um serviço específico.

Apesar das limitações inerentes à linguagem P4, foi evidenciada a viabilidade e vantagem, de desenvolver soluções independentes da intervenção do controlador, sem a necessidade de algoritmos complexos. Essas soluções capacitam o plano de dados a tomar decisões essenciais, aliviando o controlador, para mitigar desafios significativos de segurança, presentes no paradigma das redes definidas por software, como é o caso dos ataques DDoS.

6.2 Trabalho Futuro

Como trabalho futuro, será importante expandir o âmbito deste estudo para abranger uma gama mais ampla de testes e validações. Isso incluirá a realização de testes práticos do algoritmo DPSynFloodBlock em dispositivos programáveis não virtuais, distintos dos tradicionais dispositivos comerciais, bem como a análise do desempenho do algoritmo em ambientes com conjuntos de dados reais de ataques de *SYN Flood*, para avaliar a sua eficácia em situações práticas e reais, fornecendo uma validação mais robusta deste. Adicionalmente, será pertinente explorar algoritmos de entropia relacionados com a informação dos fluxos de entrada no dispositivo, bem como desenvolver um mecanismo adaptativo para ajustar os valores limites, atualmente contantes, à dinâmica da rede para o qual o algoritmo se aplica. Estas melhorias têm como propósito otimizar a fase de monitorização, promovendo uma deteção mais precisa e eficaz de ataques DDoS do tipo *SYN Flood*. O objetivo é melhorar a precisão e eficiência na identificação desses ataques, proporcionando uma resposta mais rápida e eficaz diante de potenciais ataques.

Será também pertinente explorar soluções que se apliquem ao plano de controlo e/ou ao de aplicação que possam contribuir positivamente para as limitações da solução DPSynFloodBlock.

Referências

[1] Aladaileh, M.A.; Anbar, M.; Hasbullah, I.H.; Chong, Y.W.; Sanjalawe, Y.K. Detection techniques of distributed denial of service attacks on software-defined networking controller—A review. IEEE Access 2020, 8, 143985–143995.

[2] Kreutz, D. L., Mansilha, R. B., Miers, C. C., Chervinski, J. O., Melchior, F. H., Fernandes, R., ... & Macedo, D. (2019). Minicursos da XVII Escola Regional de Redes de Computadores. Sociedade Brasileira de Computação.

[3] P4 - Open Networking Foundation, "P4", ONF. Online. 10-10-2022. URL: https://opennetworking.org/p4/.

[4] Narayanan, N.; Sankaran, G.C.; Sivalingam, K. M. Mitigation of security attacks in the SDN data plane using P4-enabled switches. In 2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 1–6 (2019).

[5] Kaur, S., Kumar, K., Aggarwal, N., & Singh, G. (2021). A comprehensive survey of DDoS defense solutions in SDN: Taxonomy, research challenges, and future directions. Computers & Security, 110, 102423.

[6] "DDoS de 3,47 Tbps registado pela Microsoft". Online. 01-08-2023. URL: https://www.cisoadvisor.com.br/ddos-de-347-tbps-registrado-pela-microsoft/.

[7] Open Networking Foundation, "Threat Analysis for the SDN Architecture", 2016. Online. 10-12-2022. URL: www.opennetworking.org,

[8] Xia, W., Wen, Y., Foh, C. H., Niyato, D., & Xie, H. (2014). A survey on software-defined networking. IEEE Communications Surveys & Tutorials, 17(1), 27-51.

[9] Haas, Z. J., Culver, T. L., & Sarac, K. (2021). Vulnerability challenges of software defined networking. IEEE Communications Magazine, 59(7), 88-93.

[10] Feferman, D. L., Mejia, J. S., Saraiva, N., & Rothenberg, C. E. (2018). Uma nova revolucao em redes: Programacao do plano de dados com p4. In Minicursos IV Escola Regional de Informática do Piauí (ERIPI), 2018. Online. 14-12-2022. URL: https://intrig.dca.fee.unicamp.br/wp-content/uploads/2018/08/Minicurso_P4_versao_final.pdf.

[11] Rahouti, M., Xiong, K., Xin, Y., Jagatheesaperumal, S. K., Ayyash, M., & Shaheed, M. (2022). SDN security review: Threat taxonomy, implications, and open challenges. IEEE Access, 10, 45820-45854.

[12] Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 103(1), 14-76.

[13] Prabha, C., Goel, A., & Singh, J. (2022, June). A survey on sdn controller evolution: A brief review. In 2022 7th International Conference on Communication and Electronics Systems (ICCES) (pp. 569-575). IEEE.

[14] Ujcich, B. E. (2020). Securing the software-defined networking control plane by using control and data dependency techniques (Doctoral dissertation).

[15] "OpenFlow Conformance Certification". Online. 08-11-2022. URL: https://www.opennetworking.org/product-certification, 2018.

[16] "RFC 6241—Network Configuration Protocol (NETCONF)". Online. 2-01-2023. URL: https://tools.ietf.org/html/rfc6241.

[17] "OpFlex: An Open Source Approach", 2014 https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731304.html.

[18] "RFC 7047-The Open vSwitch Database Management Protocol". Online. 02-01-2023. URL: https://tools.ietf.org/ html/rfc7047.

[19] "RFC 7391—Forwarding and Control Element Separation (ForCES) Protocol Extensions". Online. 12-06-2023. URL: https://tools.ietf.org/html/rfc7391.

[20] "P4Runtime". Online. 10-10-2022. URL: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html.
[21] "OpenDaylight". Online. 19-12-2022. URL: https://www.opendaylight.org/.
[22] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., ... & Parulkar, G. (2014, August). ONOS: towards an open, distributed SDN OS. In Proceedings of the third workshop on Hot topics in software defined networking (pp. 1-6).
[23] "Floodlight OpenFlow Controller (OSS)". Online. 19-12-2022. URL: https://github.com/floodlight/floodlight#Documentation-and-Support.
[24] ENISA, "Threat Landscape and Good Practice Guide for Software Defined Networks/5G Threat Landscape and Good Practice Guide for Software Defined Networks/5G About ENISA", 2015.
[25] Gupta, B. B., Perez, G. M., Agrawal, D. P., & Gupta, D. (2020). Handbook of computer networks and cyber security. Springer, 10, 978-3.
[26] Rao, S. (2015). SDN Series Part Eight: Comparison of Open Source SDN Controllers., The New Stack, 2015. Online. 02-10-2022. URL: https://thenewstack.io/sdn-series-part-eight-comparison-of-open-source-sdn-controllers/.
[27] Nayyar A., Nagrath P., Singla B. (eds.) Software Defined Networks: Architecture and Applications. Online ISBN: 9781119857921, Wiley-Scrivener, 2022. Online. 04-10-2022. URL: https://doi.org/10.1002/9781119857921.ch3
[28] Kreutz, Diego & Ramos, Fernando & Veríssimo, Paulo. (2013). Towards secure and dependable software-defined networks. HotSDN 2013 - Proceedings of the 2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. 55-60. 10.1145/2491185.2491199.
[29] Swami, R., Dave, M., & Ranga, V. (2019). Software-defined networking-based DDoS defense mechanisms. ACM Computing Surveys (CSUR), 52(2), 1-36.
[30] Durner, R., Lorenz, C., Wiedemann, M., & Kellerer, W. (2017, July). Detecting and mitigating denial of service attacks against the data plane in software defined networks. In 2017 IEEE Conference on Network Softwarization (NetSoft) (pp. 1-6). IEEE.
[31] Gao, S., Li, Z., Xiao, B., & Wei, G. (2018). Security threats in the data plane of software-defined networks. IEEE network, 32(4), 108-113..
[32] Chica, J. C. C., Imbachi, J. C., & Vega, J. F. B. (2020). Security in SDN: A comprehensive survey. Journal of Network and Computer Applications, 159, 102595.
[33] Kaur, S., Kumar, K., & Aggarwal, N. (2021). A review on P4-Programmable data planes: Architecture, research efforts, and future directions. Computer Communications, 170, 109-129.
[34] Huang, D., Chowdhary, A., & Pisharody, S. (2018). Software-Defined networking and security: from theory to practice. CRC Press.
[35] Dargahi, T., Caponi, A., Ambrosin, M., Bianchi, G., & Conti, M. (2017). A survey on the security of stateful SDN data planes. IEEE Communications Surveys & Tutorials, 19(3), 1701-1725.
[36] Zerkane, Salaheddine & Espes, David & Parc, Philippe & Cuppens, Frédéric. (2017). Vulnerability Analysis of Software Defined Networking. 10128. 97-116. 10.1007/978-3-319-51966-1_7.
[37] Yan, Z., Zhang, P., & Vasilakos, A. V. (2016). A security and trust framework for virtualized networks and software-defined networking. Security and communication networks, 9(16), 3059-3069.
[38] Shin, S., Xu, L., Hong, S., & Gu, G. (2016, August). Enhancing network security through software defined networking (SDN). In 2016 25th international conference on computer communication and networks (ICCCN) (pp. 1-9). IEEE.
[39] Liatifis, A., Sarigiannidis, P., Argyriou, V., & Lagkas, T. (2023). Advancing sdn from openflow to p4: A survey. ACM Computing Surveys, 55(9), 1-37.

[40] Alashhab, A. A., Zahid, M. S. M., Azim, M. A., Daha, M. Y., Isyaku, B., & Ali, S. (2022). A Survey of Low Rate DDoS Detection Techniques Based on Machine Learning in Software-Defined Networks. Symmetry, 14(8), 1563.
[41] Eliyan, L. F., & Di Pietro, R. (2021). DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges. Future Generation Computer Systems, 122, 149-171.
[42] Gao, Shang; Peng, Zhe; Xiao, Bin; Hu, Aiqun†; Ren, Kui, FloodDefender: Protecting Data and Control PlaneResources under SDN-aimed DoS Attacks, vol. IEEE INFOCOM 2017.
[43] Kalkan, K., Gur, G., & Alagoz, F. (2017). Defense mechanisms against DDoS attacks in SDN environment. IEEE Communications Magazine, 55(9), 175-179.
[44] Chen, X., Liu, H., Zhang, D., Huang, Q., Zhou, H., Wu, C., & Yang, Q. (2022). Empowering DDoS Attack Mitigation with Programmable Switches. IEEE Network.
[45] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. ACM SIGCOMM computer communication review, 38(2), 69-74.
[46] Wikipedia, "OpenFlow". Online. 08-11-2022. URL: https://pt.wikipedia.org/wiki/OpenFlow.
[47] Ghonge, M. M., & N, P. (2022). Software-defined network-based vehicular ad hoc networks: a comprehensive review. Software Defined Networking for Ad Hoc Networks, 33-53.
[48] "OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06)", 2015. Online. 08-11-2022. URL: http://www.opennetworking.org
[49] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., ... & Walker, D. (2014). P4: Programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review, 44(3), 87-95.
[50] Manzanares-Lopez, P., Muñoz-Gea, J.P., & Malgosa-Sanahuja, J. (2021). Passive In-Band Network Telemetry Systems: The Potential of Programmable Data Plane on Network-Wide Telemetry. IEEE Access, 9, 20391-20409.
[51] The P4 Language Consortium, "P416 Language Specification version 1.2.3". Online. 02-10-2022. URL: http://p4.org
[52] Vladimir Gurevich, P4_16 Introduction, 2017. Online. 10-10-2022. URL: https://opennetworking.org/wp-content/uploads/2020/12/p4_d2_2017_p4_16_tutorial.pdf.
[53] Hauser, F., Häberle, M., Merling, D., Lindner, S., Gurevich, V., Zeiger, F., ... & Menth, M. (2023). A survey on data plane programming with p4: Fundamentals, advances, and applied research. Journal of Network and Computer Applications, 212, 103561.
[54] "p4c". Online. 06-10-2022. URL: https://github.com/p4lang/p4c. https://github.com/p4lang/p4c.
[55] The P4 Language Consortium, "The P4 Language Specification". 2018.
[56] AlSabeh, A., Khoury, J., Kfoury, E., Crichigno, J., & Bou-Harb, E. (2022). A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment. Computer Networks, 207, 108800.
[57] Hauser, F. (2022). Integration of network security mechanisms in softwarized networks with data plane programming and software-defined networking (Doctoral dissertation, Universität Tübingen).
[58] "P4 Runtime: Future of SDN". Online. 10-10-2022. URL: https://www.volansys.com/p4-runtime-future-of-sdn/.
[59] "gRPC". Online. 24-10-2022. URL: https://grpc.io/.
[60] Hauser, F., Häberle, M., Schmidt, M., & Menth, M. (2020). P4-ipsec: Site-to-site and host-to-site vpn with ipsec in p4-based sdn. IEEE Access, 8, 139567-139586.
[61] "P4Runtime Specification version 1.3.0", P4Runtime Specification version 1.3.0, 2021. Online. 26-12-2022. URL: https://p4.org/p4-

spec/p4runtime/main/P4Runtime-Spec.html#:~:text=P4Runtime%20Specification%20version%201.3.0%20The,P4.org%20API%20Working%20Group%202021-07-02.

[62] Michel, O., Bifulco, R., Retvari, G., & Schmid, S. (2021). The programmable data plane: Abstractions, architectures, algorithms, and applications. ACM Computing Surveys (CSUR), 54(4), 1-36.

[63] Scholz, D., Stubbe, H., Gallenmüller, S., & Carle, G. (2020, September). Key properties of programmable data plane targets. In 2020 32nd International Teletraffic Congress (ITC 32) (pp. 114-122). IEEE.

[64] Gao, Ya, and Zhenling Wang. "A review of P4 programmable data planes for network security." Mobile Information Systems 2021 (2021): 1-24.

[65] Kfoury, E. F., Crichigno, J., & Bou-Harb, E. (2021). An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. IEEE access, 9, 87094-87155.

[66] Alvarez-Horcajo, J., Martinez-Yelmo, I., Lopez-Pajares, D., Carral, J. A., & Savi, M. (2021). A hybrid SDN switch based on standard P4 code. IEEE Communications Letters, 25(5), 1482-1485.

[67] Mihai Budiu, "Programming networks with P4", 2017.: Online. 10-10-2022. URL: https://octo.vmware.com/programming-networks-with-p4/1/8

[68] P4.org, "In-band Network Telemetry (INT) Dataplane Specification Version 2.1".

[69] Hang, Z., Wen, M., Shi, Y., & Zhang, C. (2019). Interleaved sketch: Toward consistent network telemetry for commodity programmable switches. IEEE Access, 7, 146745-146758.

[70] Patetta, M., Secci, S., & Taktak, S. (2022, July). A Lightweight Southbound Interface for Standalone P4-NetFPGA SmartNICs. In 2022 1st International Conference on 6G Networking (6GNet) (pp. 1-4). IEEE.

[71] "P4 Open Source Programming Language". Online. 10-10-2022. URL: https://p4.org/.

[72] "NETSCOUT DDoS Threat Intelligence Report, Findings from 1st half 2023". Online. 07-07-2023. URL: https://www.netscout.com/threatreport/key-findings/.

[73] Akamai Blog, "CVE-2022-26143: TP240PhoneHome Reflection/Amplification DDoS Attack Vector". Online. 10-10-2022. URL: https://www.akamai.com/blog/security/phone-home-ddos-attack-vector.

[74] "Mininet". Online. 27-09-2022. URL: http://mininet.org/.

[75] "Introduction to Mininet". Online. 27-09-2022. URL: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.

[76] "Iperf". Online. 21-05-2023. URL: https://iperf.fr/

[77] "Wireshark". Online. 21-05-2023. URL: https://www.wireshark.org/

[78] LinkedIn, "Practical Software-Defined Networking: 3 Learning Mininet". Online. 21-05-2023. URL: https://www.linkedin.com/learning/practical-software-defined-networking-3-learning-mininet.

[79] "Mininet Custom Topologies". Online. 21-05-2023. URL: https://www.youtube.com/watch?v=yHUNeyaQKWY.

[80] "Miniedit". Online. 21-05-2023. URL: https://github.com/mininet/mininet/blob/master/examples/miniedit.py.

[81] J. Crichigno, "P4 Programmable data planes: applications stateful elements and, custom packet processing", 2022. Online. 2-10-2022. URL: https://www.netdevgroup.com/content/usc/documentation/netlab_uofsc_p4pdp_asecpp_pod_install_guide.pdf.

[82] J. Crichigno, "Introduction to P4 Programmable Data planes", 2022. Online. 2-10-2022. URL: https://www.netdevgroup.com/content/usc/labs/intro_to_p4pdp.html

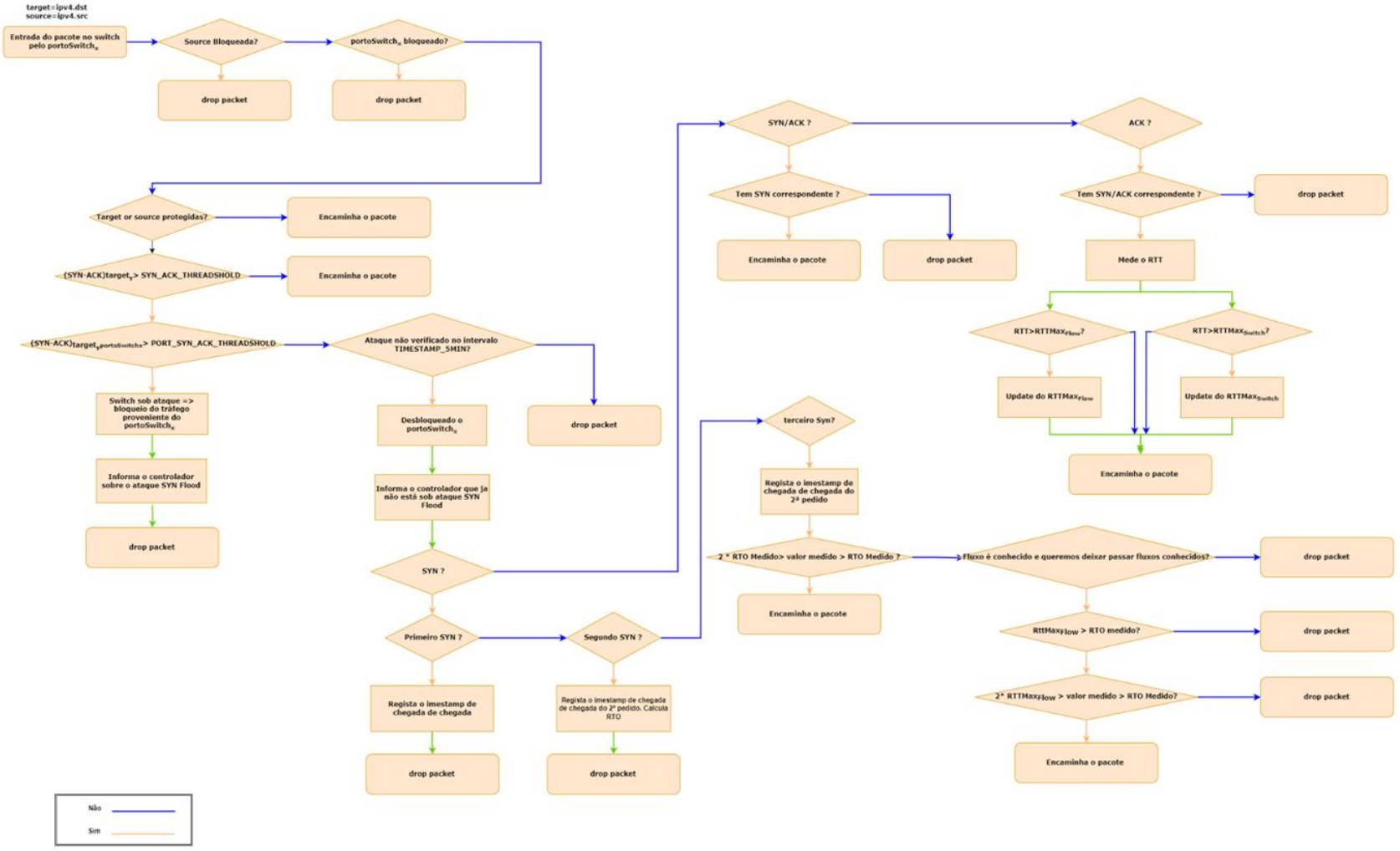
[83] "RFC 2698 - A Two Rate Three Color Marker ". Online. 02-05-2023. URL: https://www.rfc-editor.org/rfc/rfc2698.

[84] "RFC 2697 - A Single Rate Three Color Marker". Online. 02-05-2023. URL: https://www.rfc-editor.org/rfc/rfc2697.

[85] Domitilia Noro, “DPSynFloodBlock”. Online. 10-01-2024. URL: https://github.com/domnoro/DPSynFloodBlock
[86] “P4-guide”. Online. 15-10-2022. URL: https://github.com/jafingerhut/p4-guide/blob/master/bin/README-install-troubleshooting.md.
[87] “Scapy - Packet crafting for Python2 and Python3”. Online. 15-05-2023. URL: https://scapy.net/.
[88] “Hping wiki”. Online. 15-05-2023. URL: http://wiki.hping.org/
[89] “Top”. Online. 15-05-2023. URL: https://web.archive.org/web/20080413180808/http://www.unixtop.org/license.shtml
[90] GÉANT Data Plane Programmibilty, “Implementations done within the Data Plane Programmability activity of the GÉANT project”. Online. 15-05-2023. URL: https://github.com/GEANT-DataPlaneProgramming/
[91] “RFC 9293”. Online. 02-05-2023. URL: https://datatracker.ietf.org/doc/html/rfc9293
[92] “Understanding RTT Impact on TCP Retransmissions”. Online. 02-05-2023. URL: https://www.catchpoint.com/blog/tcp-rtt.
[93] “RFC 6298”. Online. 02-05-2023. URL: https://datatracker.ietf.org/doc/html/rfc6298
[94] Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms, 55(1), 58-75.
[95] Karan Shukla, “Big data with sketchy structures, part 1 — the count-min sketch”, 2018. Online. 03-06-2023. URL: https://towardsdatascience.com/big-data-with-sketchy-structures-part-1-the-count-min-sketch-b73fb3a33e2a
[96] “Probabilistic Filters by Example”. Online. 03-06-2023. URL: https://bdupras.github.io/filter-tutorial/
[97] “xxHash”. Online. 15-06-2023. URL: https://xxhash.com/
[98] Ding, D., Savi, M., & Siracusa, D. (2021). Tracking normalized network traffic entropy to detect DDoS attacks in P4. IEEE Transactions on Dependable and Secure Computing, 19(6), 4019-4031.
[99] “LAND Attacks”. Online. 26-06-2023. URL: https://www.imperva.com/learn/ddos/land-attacks/
[100] “SYN-Authentication with Bloom filter”. Online. 29-06-2023. URL: https://github.com/syn-proxy/P4-implementations/blob/master/bmv2/syn-auth-reset-bloomfilter/

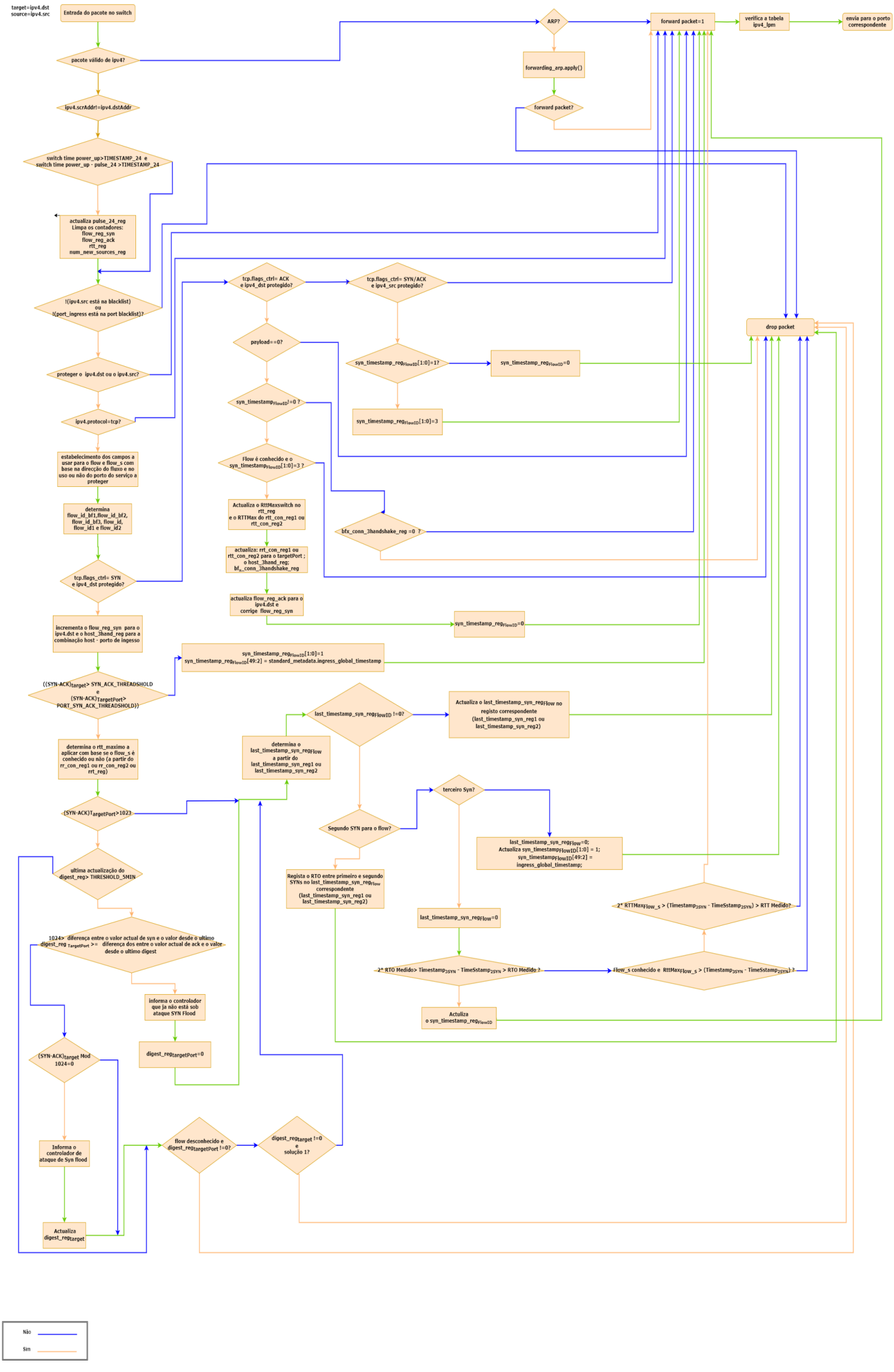
Anexo A

Fluxo de alto nível da solução DPSynFloodBlock



Anexo B

Diagrama de fluxo de DPSynFloodBlock



Anexo C

Entradas nas tabelas para testes na *testbed*

Switch s1:

MyIngress.ipv4_lpm: (default action) => MyIngress.drop()
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.1.1', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:01:11, port=1)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.2.2', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:02:22, port=2)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.3.3', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:03:00, port=3)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.4.4', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:04:00, port=4)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.5.5', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:05:00, port=3)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.1.1 => MyIngress.forward_arp(port=1)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.2.2 => MyIngress.forward_arp(port=2)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.3.3 => MyIngress.forward_arp(port=3)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.4.4 => MyIngress.forward_arp(port=4)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.5.5 => MyIngress.forward_arp(port=3)
MyIngress.target_host: hdr.ipv4.dstAddr=10.0.3.3 => MyIngress.target_host_forward(solution=1, index=3)
MyIngress.target_host: hdr.ipv4.dstAddr=10.0.4.4 => MyIngress.target_host_forward(solution=1, index=4)
MyIngress.target_host: hdr.ipv4.dstAddr=10.0.5.5 => MyIngress.target_host_forward(solution=1, index=5)
MyIngress.source_host: hdr.ipv4.srcAddr=10.0.3.3 => MyIngress.source_host_forward(solution=1, index=3)
MyIngress.source_host: hdr.ipv4.srcAddr=10.0.4.4 => MyIngress.source_host_forward(solution=1, index=4)
MyIngress.source_host: hdr.ipv4.srcAddr=10.0.5.5 => MyIngress.source_host_forward(solution=1, index=5)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=11)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=12)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=13)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=14)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=15)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=16)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=17)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=18)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=19)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=20)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=21)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=22)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=23)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=24)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=25)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=26)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=27)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=28)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=29)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=30)

Switch s2:

MyIngress.ipv4_lpm: (default action) => MyIngress.drop()
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.1.1', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:01:00, port=4)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.2.2', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:02:00, port=3)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.3.3', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:03:33, port=1)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.4.4', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:04:44, port=2)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.5.5', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:05:00, port=4)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.1.1 => MyIngress.forward_arp(port=4)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.2.2 => MyIngress.forward_arp(port=3)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.3.3 => MyIngress.forward_arp(port=1)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.4.4 => MyIngress.forward_arp(port=2)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.5.5 => MyIngress.forward_arp(port=4)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=31)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=32)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=33)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=34)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=35)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=36)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=37)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=38)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=39)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=40)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=41)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=42)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=43)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=44)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=45)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=46)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=47)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=48)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=49)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=50)

Switch s3:

MyIngress.ipv4_lpm: (default action) => MyIngress.drop()
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.1.1', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:01:00, port=1)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.2.2', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:02:00, port=1)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.3.3', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:03:00, port=2)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.4.4', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:04:00, port=2)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.5.5', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:05:55, port=3)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=51)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=52)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=53)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=54)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=55)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=56)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=57)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=58)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=59)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=60)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=61)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=62)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=63)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=64)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=65)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=66)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=67)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=68)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=69)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=70)

Switch s4:

MyIngress.ipv4_lpm: (default action) => MyIngress.drop()
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.1.1', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:01:00, port=2)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.2.2', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:02:00, port=2)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.3.3', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:03:00, port=1)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.4.4', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:04:00, port=1)
MyIngress.ipv4_lpm: hdr.ipv4.dstAddr=['10.0.5.5', 32] => MyIngress.ipv4_forward(dstAddr=08:00:00:00:05:00, port=1)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.1.1 => MyIngress.forward_arp(port=2)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.2.2 => MyIngress.forward_arp(port=2)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.3.3 => MyIngress.forward_arp(port=1)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.4.4 => MyIngress.forward_arp(port=1)
MyIngress.forwarding_arp: hdr.arp.tpa=10.0.5.5 => MyIngress.forward_arp(port=1)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=71)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=72)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=73)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.1.1, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=74)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=75)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=76)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=77)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.2.2, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=78)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=79)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=80)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=81)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.3.3, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=82)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=83)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=84)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=85)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.4.4, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=86)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=1 => MyIngress.host_portindex(pair_num=87)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=2 => MyIngress.host_portindex(pair_num=88)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=3 => MyIngress.host_portindex(pair_num=89)
MyIngress.hostportind: hdr.ipv4.dstAddr=10.0.5.5, standard_metadata.ingress_port=4 => MyIngress.host_portindex(pair_num=90)

Anexo D

Instruções usadas nos testes de comparação de algoritmos

1 : Executar numa janela de terminal na máquina virtual (disponível em [86]):
cd /home/p4/tutorials/exercises/basic_4sw_s_x
make stop
make clean
(echo ''; echo 'Date:'; date; top -b -n 2 -o %CPU) >> top_date.txt

make run
(echo ''; echo 'Date:'; date; top -b -n 2 -o %CPU) >> top_date.txt

2: Na área de Mininet executar:
xterm h1 h2 h3 s1
h1 wireshark &
h2 wireshark &
h3 wireshark &

3: Na Janela do xterm h3: -> (python3 script_top.py hh:mm:41 500 '(echo ""; echo "Date:"; date ; top -b -n 1 -o %CPU) >> top_date.txt') & python3 -m http.server 80

4: Na Janela do xterm h2:-> python3 h_comando_generico_time_limit.py (*00:02:41 + 00:01:00*) 1 "curl 10.0.3.3";sleep 180; curl 10.0.3.3; curl 10.0.3.3

7: Na Janela do xterm h1:->python3 h_comando_generico_time_limit.py (*00:02:41* + 00:01:00) 1 "date; curl 10.0.3.3"; hping3 -d 14 -S -w 64 -p 80 --flood --rand-source 10.0.3.3 & read -t 5; kill $!;python3 h_comando_generico_time_limit.py (*00:02:41 + 00:03:00*) 3 "curl 10.0.3.3"; sleep 120; python3 h_comando_generico_time_limit.py (*00:02:41* + 00:08:00) 10 "curl 10.0.3.3; sleep 10"

8: Na janela de terminal na máquina virtual:
(echo ''; echo 'Date:'; date ; top -b -n 2 -o %CPU) >> top_date.txt
fechar e salvar Wireshark de h1, h2 e h3
fechar xterm h1, h2, h3 e s1
(echo ''; echo 'Date:'; date ; top -b -n 2 -o %CPU) >> top_date.txt

9: Na área de Mininet executar:
exit

10: Na janela de terminal na máquina virtual:
(echo ''; echo 'Date:'; date ; top -b -n 2 -o %CPU) >> top_date.txt