UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

MÁRIO AUGUSTO DA CRUZ

Detecção Online de Agregações Hierárquicas Bidimensionais de Fluxos em Redes Definidas por Software

Goiânia
2014

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Detecção Online de Agregações Hierárquicas Bidimensionais de Fluxos em Redes Definidas por Software

Autor(a): Mário Augusto da Cruz

Goiânia, 16 de Dezembro de 2014.

Mário Augusto da Cruz – Autor

Dr. Kleber Vieira Cardoso – Orientador

Dra. Sand Luz Corrêa – Co-Orientadora

MÁRIO AUGUSTO DA CRUZ

Detecção Online de Agregações Hierárquicas Bidimensionais de Fluxos em Redes Definidas por Software

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Kleber Vieira Cardoso

Co-Orientadora: Profa. Dra. Sand Luz Corrêa

Goiânia
2014

MÁRIO AUGUSTO DA CRUZ

Detecção Online de Agregações Hierárquicas Bidimensionais de Fluxos em Redes Definidas por Software

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 16 de Dezembro de 2014, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Kleber Vieira Cardoso
Instituto de Informática – UFG
Presidente da Banca

Profa. Dra. Sand Luz Corrêa
Instituto de Informática – UFG

Prof. Dr. Thierson Couto Rosa
Instituto de Informática – UFG

Prof. Dr. Antônio Jorge Gomes Abelém
Faculdade de Computação – UFPA

Mário Augusto da Cruz

Graduou-se em Tecnologia em Redes de Computadores pela Faculdade de Tecnologia SENAI de Desenvolvimento Gerencial (2006-2008). No período do mestrado, atuou como integrante da equipe do GT-ATER, projeto de pesquisa e desenvolvimento financiado pela Rede Nacional de Ensino e Pesquisa (RNP). Atua como servidor público federal na área de infraestrutura e gerenciamento de redes.

Dedico esta dissertação a Deus, aos meus pais por prover as condições, e aos meus amigos, por todo o apoio e carinho.

Agradecimentos

A Deus pela graça da vida e por tudo que me proporciona.
À minha mãe, Maria Helena, por todo o amor, compreensão, apoio e paciência.
Ao Prof. Kleber Vieira Cardoso, por sua orientação, amizade, paciência e confiança.
À Profa. Sand Luz Corrêa, por sua contribuição no trabalho, orientação e apoio.
Aos Profs. Thierson Couto Rosa e Antônio Jorge Gomes Abelém, pela presença na banca e contribuições à dissertação.
Aos meus amigos de longa data: Daniel, Lincoln, Geyson, Carlos, Leo, e todos os demais; pela amizade, apoio e momentos de descontração.
Aos colegas do grupo de pesquisa Labora: Micael, Bruno, Cleber, Vivian, Lafaiet, Fausto, Otto, Pedro, e todos os demais; pela amizade, apoio e momentos de descontração.
Aos colegas do Cercomp: Jean, Douglas, Thiago, Victor, Marcello, e todos os demais; pela amizade, apoio e momentos de descontração.
À equipe da secretaria: Mirian, Patrícia e todos os demais; pela atenção, paciência e suporte operacional.
Ao INF/UFG, pelas instalações e equipamentos utilizados.
À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e à Fundação de Amparo à Pesquisa do Estado de Goiás (Fapeg), pelo suporte financeiro.

A ciência, como um todo, não é nada mais do que um refinamento do pensar diário.

Albert Einstein (1879 - 1955),
Essays in physics.

Resumo

da Cruz, Mário Augusto. **Detecção Online de Agregações Hierárquicas Bidimensionais de Fluxos em Redes Definidas por Software**. Goiânia, 2014. 88p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

As Redes Definidas por Software representam um novo paradigma que flexibiliza a operação, o monitoramento e a gerência de redes através do desacoplamento entre o plano de controle e o plano de dados. No entanto, nesse novo contexto, algumas soluções clássicas da área de monitoramento de redes precisam ser revistas, pois há novas restrições, mas também novas oportunidades. No contexto de monitoramento, uma estratégia comumente utilizada, sobretudo em redes de alta capacidade, é o acompanhamento dos itens mais frequentes, também conhecidos como *heavy hitters*. Uma das abordagens para monitoramento dos itens mais frequentes consiste em detectar as agregações hierárquicas de fluxos, a qual possibilita realizar um monitoramento eficiente em tempo real. Neste trabalho, propomos e avaliamos uma nova solução de monitoramento capaz de detectar de maneira *online* as agregações hierárquicas de fluxos, utilizando características de redes definidas por software, em especial do protocolo OpenFlow. Nossa proposta, combina uma contabilização flexível de regras de fluxos, proveniente dos comutadores OpenFlow, com uma inspeção de amostras de tráfego através de um dispositivo dedicado. Avaliamos nossa proposta em ambientes simulado e emulado, utilizando *traces* de pacotes gerados artificialmente e também de redes reais. Os resultados mostram que nossa proposta possui uma acurácia satisfatória e baixo tempo de convergência em comparação a uma solução anterior para redes OpenFlow, além de identificar *heavy hitters* em duas dimensões.

Palavras–chave

Redes Definidas por Software, OpenFlow, Monitoramento de Rede, Agregações Hierárquicas de Fluxos, Detecção de Itens Frequentes

Abstract

da Cruz, Mário Augusto. **Online Detection of Bidimensional Hierarchical Heavy Hitters in Software-Defined Networks**. Goiânia, 2014. 88p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Software Defined Networking represents a new paradigm that eases the operation, monitoring and network managing through the decoupling between the control plane and the data plane. However, in this new context, some classic solutions in the network monitoring field need to be revisited, as there are new constraints, but there are also new opportunities. In monitoring context, one strategy commonly used, mainly in high capacity networks, is the tracking of the most frequent items, also known as *heavy hitters*. One approach to monitoring the most frequent items consists in detecting the hierarchical heavy hitters, which allows an efficient real time monitoring. In this work, we propose and evaluate a new monitoring solution capable of online detection of hierarchical heavy hitters, using the characteristics of software defined networks, in special the OpenFlow protocol. Our proposal, combines a flexible accounting of flow rules, from OpenFlow switches, with inspection of traffic samples through a dedicated device. We evaluate our proposal in a simulated and emulated environments, both using packet traces generated artificially and also from real networks. The results show that our proposal has satisfactory accuracy and low convergence time in comparison to a previous solution to OpenFlow networks, in addition to identify heavy hitters in two dimensions.

Keywords

Software Defined Networking, OpenFlow, Networking Monitoring, Hierarchical Heavy Hitters, Heavy Hitter Detection

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Acrônimos

API *Application Programming Interface*. 23
ASIC *Application Specific Integrated Circuit*. 21

CLI *Command Line Interface*. 20

ForCES *Forwarding and Control Element Separation*. 29

HH *Heavy Hitter*. 17
HHH *Hierarchical Heavy Hitter*. 18
HTTP *Hypertext Transfer Protocol*. 30

IETF *Internet Engineering Task Force*. 32
IPFIX *Internet Protocol Flow Information eXport*. 35

MIB *Management Information Base*. 33

NOS *Network Operating System*. 23

ONF *Open Networking Foundation*. 23

PSAMP *Packet SAMPling*. 32

SDN *Software Defined Networking*. 16
SNMP *Simple Network Management Protocol*. 33

TCAM *Ternary Content Addressable Memory*. 26
TCP *Transmission Control Protocol*. 32
TM *Traffic Matrix*. 31

VLAN *Virtual Local Area Network*. 24

CAPÍTULO 1

Introdução

1.1 Motivação

Atualmente, muitos usuários dependem do acesso à Internet da mesma forma como dependem de alguns serviços básicos como energia elétrica ou água potável. Nesse contexto, é crescente a pressão sobre provedores de Internet para garantir a disponibilidade de infraestruturas e serviços oferecidos. Comumente, a manutenção da infraestrutura exige acompanhar o funcionamento de seus componentes ativos (por exemplo, comutadores e roteadores) através de medições regulares. Essas medições contêm informações importantes para determinar problemas ou questões de desempenho, como a existência de enlaces congestionados ou subutilizados, a necessidade de troca de equipamentos, ou quais os horários do dia em que os enlaces estão com maior ocupação.

De maneira geral, o processo de realizar medições em pontos específicos de uma rede, de modo que se possa inferir um comportamento normal ou não sobre o estado atual da rede, é conhecido por monitoramento. Os processos de monitoramento buscam detectar eventos que possam comprometer o funcionamento e o desempenho da rede, por exemplo: falhas de configuração em políticas de roteamento, comportamento intermitente de equipamentos, interrupção de enlaces, ataques distribuídos de negação de serviço (DDoS), etc.

As ferramentas de monitoramento devem ser capazes de se adaptar às características dinâmicas do tráfego de pacotes, interferir o mínimo possível nos enlaces e ativos observados, e reportar comportamentos ou eventos que fogem do normal. Tudo isso deve ser realizado no menor tempo possível. Essas ferramentas também devem oferecer estatísticas precisas acerca do tráfego, uma vez que essas informações devem ser usadas para auxiliar a tomada de decisões do operador de rede.

O monitoramento de redes é um tema amplamente estudado na literatura, apresentando diversas soluções. No entanto, um novo paradigma, conhecido como Redes Definidas por Software [28, 54, 55] (*Software Defined Networking* - SDN) vem demandando uma revisita ao tema. Dentre as motivações de SDN, estão a dificuldade para inovação em redes tradicionais, alta verticalização das soluções com base em hardware e software

proprietários e complexidade para realizar atualização ou reconfiguração de ativos de redes que normalmente são oferecidos como sistemas embarcados. Nesse contexto, SDN propõe o desacoplamento entre o plano de controle, que é a camada de software responsável pelo controle, gerência e monitoramento nos ativos de rede, e o plano de dados, que é a camada de hardware responsável pelo encaminhamento dos pacotes [52]. Além desse desacoplamento, as redes SDN possuem um componente central chamado de controlador, o qual é responsável pela configuração da tabela de encaminhamento dos fluxos nos dispositivos de rede. Esses dispositivos podem ser qualquer ativo de rede que realize algum processamento de pacote, por exemplo, comutador, roteador, NAT, *firewall*, etc.

Em uma rede SDN, o controlador pode realizar o controle, a gerência e o monitoramento em diversos ativos de rede, independente do fabricante do equipamento e, portanto, rompendo a integração vertical que havia nas redes convencionais. Através da centralização, o administrador, a partir de único ponto, pode monitorar o estado global da rede, assim como aplicar novas políticas sem que haja a necessidade de efetuar a reconfiguração manual de diversos equipamentos. Outra vantagem das redes SDN é que o controlador foi projetado com interfaces abertas e programáveis, ou seja, o administrador pode desenvolver uma aplicação no controlador para disponibilizar novas funcionalidades na rede, por exemplo, um protocolo de roteamento diferente, uma nova abordagem para monitoramento ou um sistema inovador para detecção de anomalias.

O protocolo OpenFlow (OF) [55] é uma das primeiras iniciativas práticas de SDN. Essa iniciativa teve início na Universidade de Stanford e tem recebido significativa atenção, tanto da academia quanto da indústria. O protocolo OF define um conjunto de instruções que são mapeadas diretamente no plano de dados do dispositivo como uma regra de fluxo. Cada regra de fluxo possui uma ação associada. Dessa forma, o controlador pode realizar várias operações com os fluxos que entram no comutador, por exemplo: descartar pacotes, enviar para o controlador, enviar uma cópia dos pacotes para uma ou mais portas de saída. O controlador, utilizando uma combinação de regras, pode fazer o comutador se comportar como um *firewall* ou um dispositivo de detecção de intrusão, por exemplo.

Uma estratégia de monitoramento que merece destaque, especialmente em redes de alta capacidade, é a que se concentra no acompanhamento dos itens mais frequentes. Esses itens correspondem a agregações de tráfego de pacotes que são conhecidas como *heavy hitters* (HHs) [16] ou fluxos elefantes [27]. É natural o interesse por esse tipo de fluxo ou agregação, pois uma pequena quantidade de *heavy hitters* é capaz de consumir uma grande quantidade de recursos da rede [98].

O monitoramento de *heavy hitters* consiste em encontrar os fluxos de pacotes cujo volume esteja acima de um determinado limiar de detecção. É comum que o tráfego de rede possa ser agregado com base em hierarquias. O tráfego de pacotes IP

é um exemplo de fluxo de dados que pode ser representado de maneira hierárquica. A generalização hierárquica do monitoramento de HHs é chamada de *Hierarchical heavy hitters* (HHHs). Basicamente, os HHHs são os *heavy hitters* que estão dispostos em uma dada estrutura hierárquica, oferecendo uma informação mais precisa sobre as agregações de tráfego que mais consomem recursos da rede.

O principal objeto de investigação desta dissertação é propor uma nova solução de monitoramento capaz de detectar de maneira *online* os HHHs, utilizando as características de Redes Definidas por Software, em especial os recursos do protocolo OpenFlow. A solução proposta é avaliada e comparada com outra solução para redes OpenFlow [49], apresentando resultados superiores de desempenho além de funcionalidades adicionais, como tratamento bidimensional do tráfego e escalabilidade.

1.2 Objetivos

O principal objetivo dessa dissertação é apresentar uma solução para detecção *online* das agregações hierárquicas de fluxos, as quais são responsáveis pelo consumo da maior parte dos recursos da rede. Nossa solução foi projetada para operar em redes OpenFlow e é composta por duas partes principais. Uma dessas partes é uma aplicação OpenFlow, executada em um controlador OF, sendo responsável por interagir com os comutadores, obter estatísticas dos fluxos de pacotes, identificar e reportar HHHs. Essa primeira parte da solução utiliza um método interativo para detectar HHHs e, em geral, apresenta um tempo consideravelmente longo para convergência. Por essa razão, a solução possui outra parte capaz de lidar com amostras de tráfego e identificar mais rapidamente as agregações hierárquicas. No entanto, como essa segunda parte é computacionalmente intensa e o volume de dados que precisa ser tratado é proporcional à velocidade da rede, sua execução precisa ser planejada apropriadamente. Nesse contexto, os principais objetivos específicos desta dissertação são:

1. apresentar o projeto e a implementação da solução proposta, detalhando suas duas partes ou componentes principais;
2. descrever a validação e avaliação da solução proposta, as quais se basearam em simulação e emulação, usando tanto *traces* de tráfego sintético quanto *traces* de tráfego real;
3. mostrar a comparação entre a solução proposta e outra abordagem da literatura para redes definidas por software, descrevendo as diferenças entre as abordagens e os ganhos obtidos.

1.3 Estrutura da Dissertação

O restante da dissertação está organizada da seguinte forma:

- No capítulo 2, serão abordados os principais fundamentos teóricos utilizados neste trabalho, os quais compreendem algumas características de SDN relevantes para o contexto e também do protocolo OpenFlow. Serão mostradas as vantagens e desvantagens das atuais técnicas utilizadas para efetuar o monitoramento em enlaces de rede. Também serão apresentados os conceitos relacionados ao monitoramento dos itens mais frequentes em uma rede, os trabalhos relacionados ao monitoramento de tráfego em redes definidas por software e também soluções anteriores para detecção de itens frequentes, tanto em redes convencionais quanto em redes OpenFlow;
- No capítulo 3, serão mostrados os principais componentes que fazem parte da solução proposta, assim como os algoritmos desenvolvidos;
- No capítulo 4, serão apresentados o ambiente para avaliação dos algoritmos propostos, as métricas utilizadas nas avaliações, uma análise acerca do impacto da amostragem no funcionamento da solução, a comparação com trabalhos anteriores da literatura e também o desempenho da solução proposta;
- O capítulo 5 discutirá as principais contribuições da dissertação e também as perspectivas de trabalhos futuros.

CAPÍTULO 2

Fundamentos e Trabalhos Relacionados

Neste capítulo são apresentados conceitos, protocolos, mecanismos e tecnologias importantes no contexto da dissertação. Inicialmente, na Seção 2.1, apresentamos os conceitos relacionados às Redes Definidas por Software, destacando as vantagens desse novo paradigma em comparação com as redes convencionais. A seguir, na Seção 2.2, introduzimos uma visão geral da área de monitoramento em redes e as tecnologias e ferramentas empregadas para monitorar as redes convencionais. Ainda na Seção 2.2, descrevemos algumas iniciativas relevantes para o monitoramento em redes OpenFlow. Por fim, na Seção 2.3, apresentamos as definições relacionadas a identificação de itens frequentes hierárquicos em fluxos de pacotes e também comentamos sobre a literatura relacionada à detecção de itens frequentes tanto em redes convencionais quanto em redes OpenFlow.

2.1 SDN e OpenFlow

Em suma, os equipamentos que mantêm a infraestrutura da Internet são compostos de comutadores e roteadores, de diferentes capacidades de acordo com o tamanho da rede que eles interconectam. Em redes de médio e grande porte existem ainda dispositivos que desempenham funções específicas, conhecidos como *appliances* ou *middleboxes*, por exemplo: *firewall*, sistemas de detecção de intrusão, filtragem de email, antivírus, balanceamento de carga, etc. Basicamente, os comutadores e roteadores são compostos pelo plano de controle e pelo plano de dados.

O plano de controle é responsável pelo controle, gerência e monitoração do hardware [52]. Normalmente, o plano de controle é implementado na forma de um sistema operacional proprietário, em geral na forma de um *firmware*. É nesse sistema em que são executados os protocolos de roteamento e monitoramento, por exemplo. É também no plano de controle que o administrador de rede efetua a gerência e operação dos equipamentos, geralmente na forma de uma interface de linha de comando (CLI – *Command Line Interface*).

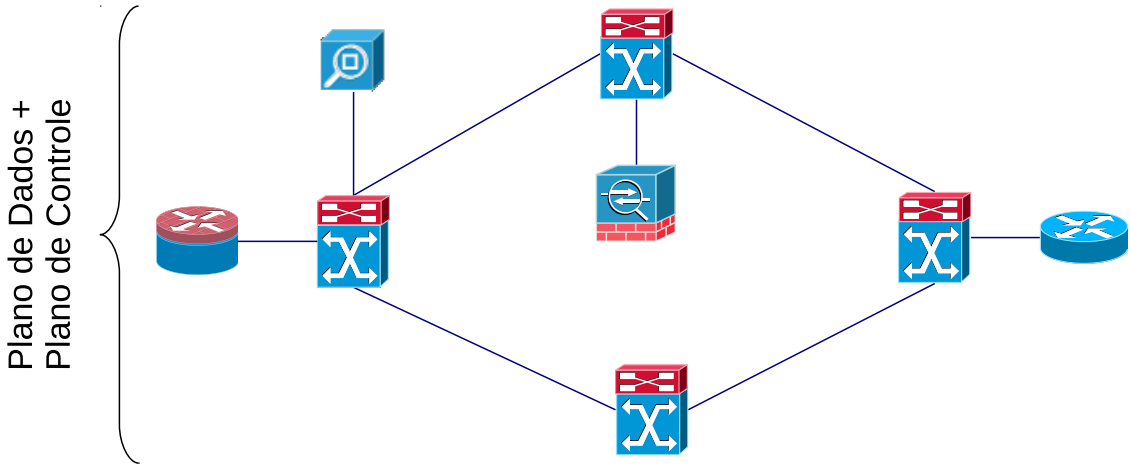
O plano de dados é responsável pelo tratamento e encaminhamento dos pacotes entre as portas de entrada e saída do equipamento. Essa parte do equipamento geralmente é composta por circuitos de hardware dedicado (ASICs – *Application Specific Integrated Circuits*) e, portanto, o plano de dados garante uma baixa latência no tratamento dos fluxos e alto desempenho no encaminhamento de pacotes.

Uma restrição para o operador de rede vem do fato de que, normalmente, os planos de controle e dados são fornecidos pelo mesmo fabricante, o qual oferece uma quantidade restrita de funcionalidades no equipamento. Outra dificuldade advém do fato de que equipamentos similares, mas de fabricantes diferentes possuírem diferentes interfaces de gerência.

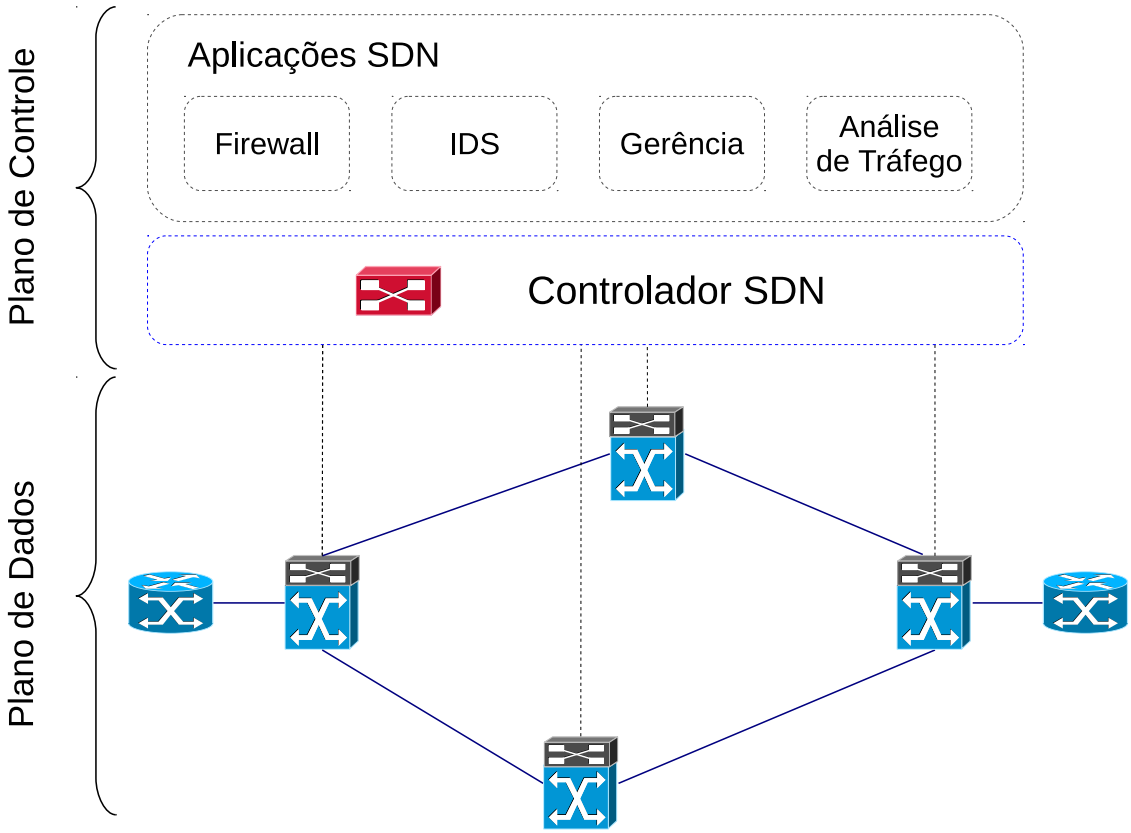
O contexto dominante das redes atuais criou a necessidade de um arcabouço de software e de hardware que possa atingir os seguintes objetivos: 1) facilitar a reconfiguração de ativos de rede, diminuindo o tempo gasto e a possibilidade de erros; 2) promover uma centralização lógica do plano de controle, de modo que a aplicação de novas políticas de rede seja transparente mesmo considerando equipamentos de fabricantes distintos; 3) tornar a rede mais resiliente a eventos que comprometem a disponibilidade, de forma que as medidas necessárias para reestabelecer seu funcionamento sejam colocadas em curso no menor tempo possível.

As Redes Definidas por Software (termo original em inglês *Software Defined Networking* – SDN) representam um novo paradigma que promete enfrentar a maioria dos problemas encontrados nas redes convencionais. Para isso, o paradigma SDN promove um desacoplamento entre o plano de controle e o plano de dados [67], possibilitando ainda que o plano de controle possa controlar vários dispositivos na rede. Operacionalmente, os equipamentos recebem comandos a partir do plano de controle e os executam ou aplicam diretamente nas tabelas de fluxos do plano de dados. No paradigma SDN, a entidade responsável por enviar os comandos de modificação da tabela de fluxos é o controlador. Esse desacoplamento simplifica a operação desses ativos uma vez que seu componente de "inteligência", ou seja, o plano de controle é transferido para o controlador, fazendo com que o plano de dados seja responsável apenas pela tarefa de encaminhamento dos pacotes. A Figura 2.1 ilustra a arquitetura básica de SDN em comparação a de uma rede convencional.

Um aspecto que fica evidente nessa abordagem é a centralização do controle da rede, com uma visão globalizada dos comutadores e roteadores, trazendo ao operador a possibilidade de aplicação de novas políticas para o tratamento dos fluxos a partir de um ponto central, ao contrário das redes convencionais. Embora o controlador possa ser visto como uma entidade logicamente centralizada, isso não é algo desejável em termos de disponibilidade da rede, pois pode introduzir um ponto único de falha. Para suprir essa deficiência, alguns trabalhos anteriores [90] [94] propõem uma arquitetura



(a) Em redes convencionais, a gerência é realizada de maneira distribuída, e algumas funções específicas são tratadas por meio de *appliances*.



(b) Nas redes SDN, o plano de dados realiza apenas o papel de encaminhamento de pacotes, enquanto que as funções de gerência, monitoramento e também aquelas desempenhadas pelos *appliances* são exercidas pelo controlador.

Figura 2.1: *Arquitetura de uma rede SDN e também de uma rede convencional.*

com múltiplos controladores fisicamente distribuídos, mas ainda mantendo a ideia de um plano de controle logicamente centralizado.

Outro aspecto importante de SDN é a implementação de padrões abertos para as interfaces de software no plano de controle. Assim, a lógica de execução no plano de controle permanece inalterada mesmo tendo equipamentos de diferentes fabricantes no plano de dados. Essas interfaces permitem que o operador possa desenvolver aplicações no plano de controle de acordo com as necessidades operacionais da rede, conforme mostrado na Figura 2.1b. A API (*Application Programming Interface*) do plano de controle estabelece uma forma padronizada para a execução de operações entre o controlador e os ativos da rede, como por exemplo: consultar a tabelas de fluxos, instalar e remover regras de encaminhamento, consultar o estado das interfaces de rede, recuperar estatísticas de regras instaladas, entre outras. Através da combinação dessas operações é possível escrever aplicações no controlador, tais como: filtro de pacotes, detecção de intrusão, monitoramento e balanceamento de carga. Devido ao fato de que múltiplas aplicações podem executar simultaneamente no controlador, esse também é conhecido pelo termo Sistema Operacional de Rede ou NOS (*Network Operating System*) [32, 52].

Em 2011, com o intuito de direcionar os esforços para a adoção de SDN na indústria de equipamentos de rede, as empresas Deutsche Telekom, Facebook, Google, Microsoft, Verizon e Yahoo criaram a *Open Networking Foundation* (ONF) [67], uma organização não lucrativa com o propósito de estudar os requisitos de arquitetura, discutir e propor a evolução tecnológica dos padrões técnicos e difundir, junto à indústria, os benefícios na utilização de SDN. Até o ano de 2014, a ONF contava com mais de 100 membros [66], com destaque para provedores de infraestrutura de telecomunicações, fabricantes de processadores e equipamentos para redes corporativas, companhias que atuam na área de centros de dados e também algumas empresas iniciantes de software com produtos voltados para o mercado de SDN.

2.1.1 OpenFlow

Desde o seu projeto, o protocolo OpenFlow já contempla a abordagem de separação dos planos de controle e de dados, de forma que o comutador realize apenas a função de encaminhamento de pacotes. Assim, o comutador passa a ser controlado através de um elemento externo chamado de controlador, o qual assume o papel do plano de controle. Na arquitetura de um comutador OpenFlow, a troca de mensagens entre o controlador e o comutador é realizada através de um canal seguro, como pode ser visto na Figura 2.2.

As instruções enviadas pelo controlador são mapeadas diretamente em regras de encaminhamento e instaladas na tabela de fluxos dos comutadores. Dessa forma, uma

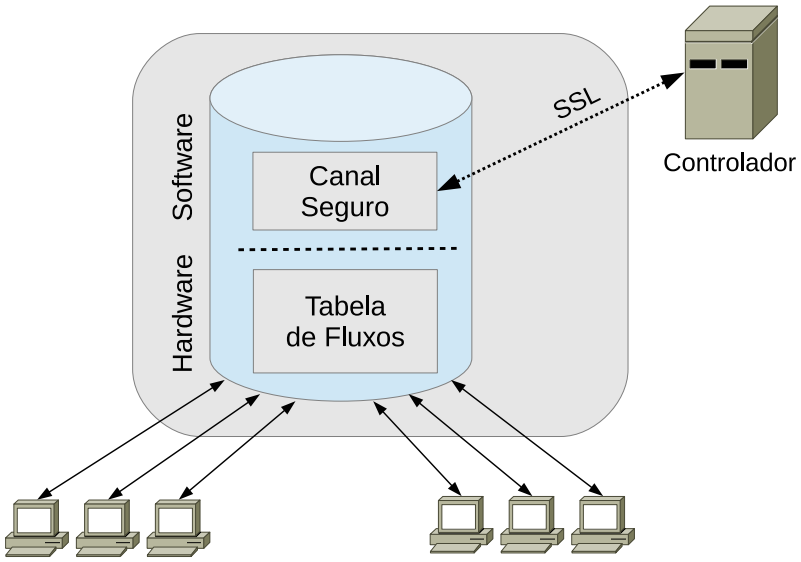


Figura 2.2: *Arquitetura de um comutador OpenFlow.*

regra instalada pelo controlador indica qual deve ser o tratamento a ser aplicado aos pacotes pelo comutador. A tabela do comutador pode conter uma ou mais entradas, sendo que cada entrada da tabela é definida por três campos: regra, contadores e ação, conforme é mostrado na Figura 2.3. O campo regra especifica algumas características do cabeçalho que são utilizadas para verificar se há uma combinação entre a entrada da tabela e o pacote que ingressa no comutador. O campo contadores armazena a quantidade de bytes e de pacotes que combinaram com a regra, além do tempo decorrido desde que a regra foi instalada. O campo ação descreve uma ou mais operações que o comutador deve realizar sobre os pacotes. As ações podem ser: encaminhar o pacote para uma ou mais portas, descartar o pacote, enviar o pacote para o controlador, alterar o endereço MAC de destino, configurar o rótulo de VLAN, aplicar parâmetros de qualidade de serviço, entre outros, conforme ilustrado na Tabela 2.1. Adicionalmente, em cada entrada da tabela existe um campo de prioridade associado à regra. Esse campo serve para que o equipamento possa distinguir entre duas ou mais regras com características semelhantes.

A cada pacote que chega, o comutador verifica se existe uma regra, dentre as que estão instaladas, que combina com as informações de cabeçalho daquele pacote. Caso haja, o comutador realiza as ações que estão descritas na regra. Caso não exista nenhuma regra instalada que combine com o pacote, o comutador gera um evento chamado *packet_in* e o envia, juntamente com o pacote, ao controlador. O controlador, ao receber o evento *packet_in*, decide, de acordo com a sua lógica interna, como deve proceder. A decisão pode ser específica para o pacote ou gerar uma nova regra que será aplicada a futuros pacotes com alguma característica em comum.

No controlador pode ser utilizada tanto uma abordagem reativa quanto uma abordagem proativa para a instalação das regras no comutador OF. Na abordagem reativa,

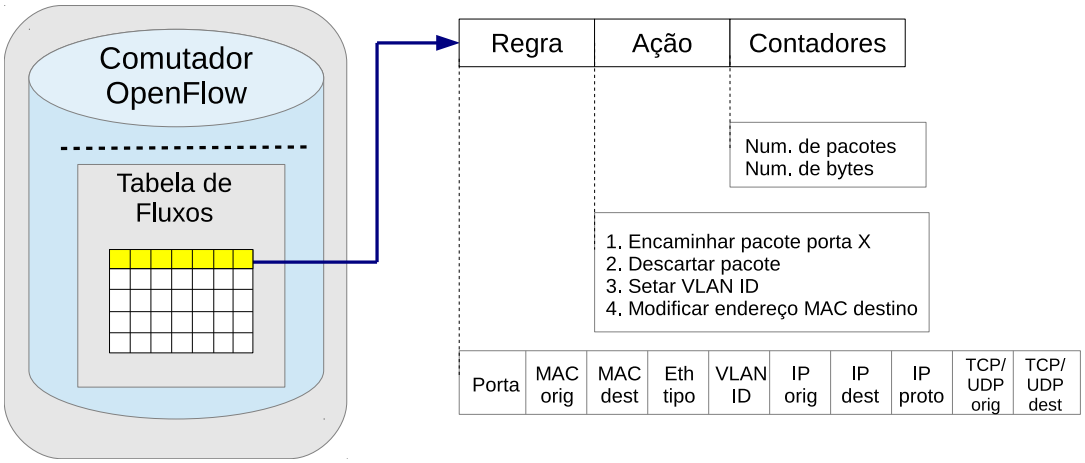


Figura 2.3: *Detalhes de uma regra OpenFlow.*

Camada	Operação	Exemplo
1	encaminhamento normal	encaminhar pacote para as portas 3 e 4
	descartar pacote	–
2	VLAN ID	adicionar ou alterar ID de VLAN para 1002
	prioridade VLAN	configurar o campo prioridade VLAN para 10
	remover cabeçalho VLAN	–
	endereço MAC origem	modificar endereço MAC de origem para 00:16:3E:49:8B:99
	endereço MAC destino	modificar endereço MAC de destino para 00:16:3E:56:1F:2C
3	endereço IP origem	modificar endereço IP de origem para 192.168.1.120
	endereço IP destino	modificar endereço IP de destino para 192.168.155.90
	flags IP ToS	modificar flags IP ToS para 101
4	protocolo transporte porta origem	modificar protocolo transporte porta de origem para 80
	protocolo transporte porta destino	modificar protocolo transporte porta de destino para 22

Tabela 2.1: *Exemplos de ações que podem ser associadas a uma regra OpenFlow.*

o controlador efetua apenas a instalação de regras em resposta a eventos *packet_in*. Uma desvantagem dessa abordagem é a dependência em relação ao tempo de resposta do controlador, uma vez que poderiam ocorrer atrasos para a instalação das regras caso o controlador estivesse com uma alta carga de processamento. Outro problema está relacionado à quantidade de eventos *packet_in* que o controlador consegue tratar, pois em uma rede com alta vazão podem trafegar milhares de fluxos por segundo. Utilizando a abordagem proativa, o controlador efetua a instalação de regras previamente a existência dos fluxos. A principal vantagem dessa abordagem é o encaminhamento eficiente, pois o

comutador não precisa aguardar o processamento de eventos *packet_in*.

Comutadores OF implementados em hardware possuem uma quantidade limitada de entradas na tabela de fluxos. A principal razão para essa limitação é o alto custo da memória utilizada, a qual é chamada TCAM (*Ternary Content Addressable Memory*). Além do custo elevado, cerca de 80 vezes o custo da memória SRAM [48], a TCAM também apresenta alto consumo energético. Logo, é importante fazer um uso eficiente das regras instaladas na tabela de fluxos.

Há duas abordagens para manutenção de regras a partir do controlador: remoção explícita e expiração automática. A especificação do protocolo OpenFlow 1.0 [24] define duas formas de expiração automática de regras: por *idle-timeout* e por *hard-timeout*. Com *idle-timeout*, a regra expira após um tempo pré-configurado de inatividade, ou seja, um tempo decorrido sem haver pacotes que combinam com a regra. Com *hard-timeout*, a regra expira após um tempo decorrido da instalação da regra, independente de haver ou não pacotes que combinam com a regra. As duas formas de expiração automática podem ser utilizadas simultaneamente, sendo que existe a opção do comutador notificar o controlador sobre a remoção de regras expiradas.

Em versões mais recentes do protocolo OpenFlow o comutador pode utilizar o recurso de múltiplas tabelas. Esse recurso permite que os pacotes sejam tratados por uma série de tabelas de fluxos encadeadas, onde cada tabela atribui um conjunto de ações ao pacote, conforme mostrado na Figura 2.4. Ao ingressar no comutador o pacote é sempre processado pela tabela 0, sendo que inicialmente o conjunto de ações está vazio. Quando existe uma correspondência entre uma regra OF da tabela e um pacote, esse pode ser encaminhado para a próxima tabela através da instrução *Goto*. Assim que o processamento de uma tabela termina, o conjunto de ações associado ao pacote é incrementado com as instruções contidas na regra OF correspondente. O processo se repete quando o pacote ingressa na tabela seguinte. Caso uma regra OF não inclua a instrução *Goto*, o processamento em múltiplas tabelas é encerrado e o conjunto de ações é executado sobre o pacote. O recurso de múltiplas tabelas foi incorporado nativamente no protocolo OpenFlow a partir da versão 1.1 [25], mas existem alguns controladores que suportam essa funcionalidade através de um conjunto de extensões da Nicira [77], o qual adiciona alguns recursos à versão 1.0 do protocolo. O processamento em múltiplas tabelas é um recurso opcional, ou seja, as aplicações OF que estão rodando no controlador podem fazer uso ou não dessa funcionalidade.

As versões 1.0 e 1.1 do protocolo foram desenvolvidas pelo mesmo grupo de pesquisadores de Stanford que propôs inicialmente o OpenFlow. A partir de 2011, as atividades de padronização e desenvolvimento de novas versões do protocolo passaram a ser de responsabilidade da *Open Networking Foundation* (ONF). Após pouco tempo, a ONF publicou a versão 1.2 do protocolo, o qual se encontra na versão 1.4. A Tabela 2.2

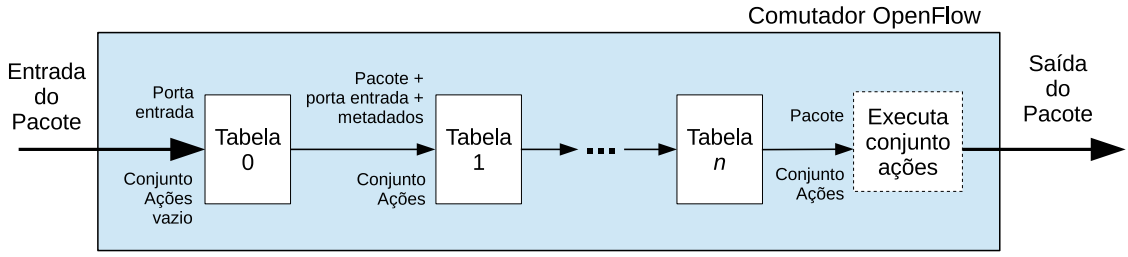


Figura 2.4: *Funcionamento do recurso de múltiplas tabelas.*

resume os principais recursos adicionados nessas novas versões.

Versão	Data	Recursos
v1.1 [25]	28/02/2011	suporte a múltiplas tabelas adição do recurso de grupo suporte extendido para rótulos de VLAN, e adição de QinQ suporte a rótulos MPLS
v1.2 [68]	05/12/2011	regras flexíveis com OXM (*OpenFlow Extensible Match*) melhorias nos metadados do evento *packet_in* suporte inicial ao protocolo IPv6 adição de mecanismos para suportar múltiplos controladores
v1.3 [69]	13/04/2012	melhorias na etapa de negociação do protocolo adição de uma regra específica de *table miss* extensão no tratamento dos cabeçalhos IPv6 adição do recurso de medidores associados a regras de fluxos suporte para adicionar rótulos PBB (*Provider Backbone Bridging*)
v1.4 [70]	14/10/2013	adição de suporte a portas ópticas adição do recurso de monitoramento das regras de fluxo mecanismos para tratar eventos de tabela cheia melhorias no envio de códigos de erro

Tabela 2.2: *Alguns recursos adicionados nas novas versões OpenFlow.*

Controladores

Como pôde ser observado, os controladores são componentes essenciais da tecnologia OpenFlow, sendo geralmente ofertados na forma de pacotes de desenvolvimento de software que implementam a API OF para troca de mensagens entre o comutador e o controlador. O NOX [32] foi o primeiro controlador OF disponível, sendo inicialmente desenvolvido pela empresa Nicira Networks e, posteriormente, tendo o código-fonte doado à comunidade acadêmica. O NOX provê uma API compatível com a especificação OF 1.0, uma interface assíncrona de entrada e saída, e exemplos de aplicações como: descoberta de topologia, comutador de camada 2, entre outros. Desenvolvido pela mesma equipe do

NOX, o controlador POX [77] possui algumas características interessantes como: a utilização de componentes reusáveis (por exemplo: componentes para decodificar protocolos do nível de aplicação, efetuar descoberta de topologia), uma API OF de fácil utilização, execução em múltiplas plataformas e componentes que possibilitam o uso das extensões OF da Nicira, como suporte à instalação de regras em múltiplas tabelas.

Apesar do protocolo OpenFlow trazer o recurso de programabilidade às redes atuais, ele não torna essa tarefa mais fácil [30]. De fato, os controladores OF tradicionais disponibilizam uma API que simula o comportamento de baixo nível presente no hardware dos comutadores. No entanto, as aplicações desenvolvidas nesses controladores manipulam estados individualmente nos equipamentos, cabendo ao programador a tarefa de sincronizar essas informações no controlador, a fim de obter um estado global da rede. Nesse contexto, foram criadas algumas linguagens de alto nível que trazem uma série de abstrações para facilitar o desenvolvimento de aplicações que interagem com vários dispositivos simultaneamente. Na linguagem Frenetic [31], por exemplo, é possível efetuar tarefas como: consultar o estado global da rede, definir políticas de encaminhamento, e efetuar reconfigurações na rede de maneira consistente entre os dispositivos. A linguagem Procera [89] tem o foco voltado para o desenvolvimento de aplicações que possuem um aspecto reativo, permitindo uma resposta rápida tanto a eventos internos gerados pelos comutadores OF, quanto a eventos externos originados em outros dispositivos, como sistemas de detecção de intrusão. A linguagem FatTire [79] possui um conjunto de construções que permite descrever e aplicar políticas de tolerância a falhas em uma rede com vários dispositivos OF.

Ferramentas de Apoio e Aplicações

Com o intuito de facilitar a prototipação e validação de novas aplicações de maneira isolada, foram criadas algumas ferramentas que simulam redes virtuais OpenFlow. O Mininet [53] foi a primeira ferramenta a possibilitar uma maneira rápida e fácil de testar aplicações OF. Com a utilização de comutadores baseados em software, essa ferramenta oferece um ambiente emulado similar a um ambiente real com dispositivos físicos. Em sua versão mais recente, o Mininet-HiFi [35] traz algumas melhorias em relação ao primeiro Mininet. Houve uma melhoria substancial de desempenho no mecanismo de emulação baseado em *container*, permitindo a criação de topologias de rede mais complexas, com centenas de *hosts* virtuais e dezenas de comutadores. Foi adicionada ao Mininet-HiFi a possibilidade de utilizar abstrações de software para executar totalmente um experimento de rede, por exemplo, criação da topologia da rede virtual, injeção de um tráfego de pacotes, modificação de parâmetros de interesse como atraso ou perda de pacotes e extração dos resultados do experimento. Já o simulador fs-sdn [33] tem o objetivo de facilitar o desenvolvimento de aplicações no controlador. Para isso, ele incorpora no simulador o

controlador POX [77] e sua API, além de componentes que simulam um comutador OF. A ferramenta EstiNet [91] combina tanto as vantagens de simulação quanto da emulação possibilitando que uma rede simulada tenha um relógio próprio, o qual pode ou não acompanhar o tempo real da máquina.

Vale a pena destacar algumas aplicações OF de propósito específico, como o RouteFlow [60, 81] que é uma plataforma para fornecimento de serviços legados de roteamento IP em redes OF. Para isso, o RouteFlow utiliza uma rede virtualizada composta por máquinas virtuais que são interconectadas dinamicamente para simular a mesma topologia de roteadores da rede real. Em cada uma dessas máquinas virtuais é executado um software de roteamento, sendo que as decisões de roteamento desse ambiente virtual são aplicados na rede real por meio de regras de fluxo nos comutadores OF. A solução Hedera [1] propõe um sistema de escalonamento dinâmico de fluxos para topologias compostas por múltiplas camadas de comutadores, como as encontradas em centros de dados. A Hedera coleta as informações de fluxos a partir dos comutadores da rede, calcula os melhores caminhos para os fluxos e aplica as regras nos equipamentos, reencaminhando o tráfego de acordo com os novos caminhos. A ElasticTree [36] é uma solução para adaptar o consumo de energia de uma rede de centro de dados. Para isso, a ElasticTree calcula o conjunto mínimo de elementos da rede que satisfaz as condições atuais de tráfego. Todos os demais elementos que estão fora do conjunto mínimo são desativados, possibilitando assim uma economia de energia nos ativos de rede.

Além das aplicações citadas, existem outros trabalhos na literatura empregando OpenFlow em áreas como: gerenciamento de redes [51], engenharia de tráfego [34, 46, 78], mobilidade em redes sem fio [85, 93], medição e monitoramento de tráfego [87, 95], detecção de ataques e segurança [6, 76, 82] e redes para centros de dados [2, 5, 23]. Uma relação extensa de aplicações para redes OpenFlow/SDN pode ser encontrada em [19, 40, 52, 54, 65].

2.1.2 Outras Abordagens

Os princípios em torno de SDN representam uma evolução de várias propostas anteriores que tinham o objetivo de trazer flexibilidade e simplificação na operação e no gerenciamento das redes [28]. O ForCES (*Forwarding and Control Element Separation*) [44] foi umas das primeiras iniciativas no sentido de propor a separação entre plano de controle e o plano de dados. Sua arquitetura utiliza um protocolo que padroniza a troca de informações entre o elemento de controle e o elemento de encaminhamento (plano de dados). Esse protocolo funciona em um modelo do tipo mestre-escravo, no qual os elementos de encaminhamento agem como escravos e os elementos de controle são os mestres. Adicionalmente, esse protocolo inclui comandos para envio de informações

de configuração, associação, estado atual do dispositivo, além de suporte a eventos e notificações.

Outra iniciativa é o Ethane [10], o qual consistia em uma arquitetura de controle centralizado da rede baseada em três princípios simples: a rede deveria ser gerenciada com o uso de políticas de alto nível, a política deveria indicar qual o caminho os pacotes devem seguir e a rede deveria ter a possibilidade de rastrear a origem dos pacotes. Para atender a esses princípios, o Ethane emprega um controlador que utiliza uma política global da rede, na qual quaisquer comunicações entre equipamentos devem ser explicitamente permitidas pelo controlador. Dessa forma, o controlador pode calcular as rotas para os fluxos permitidos, utilizando informações acerca da topologia da rede.

2.2 Monitoramento

O monitoramento é um processo que emprega um conjunto de técnicas para: 1) obtenção de informações de tráfego utilizando protocolos bem definidos; 2) agregação dessas informações provenientes de vários equipamentos diferentes; 3) análise desses dados tentando identificar falhas, mudanças abruptas no perfil de tráfego e comportamentos ou eventos que fogem do normal (geralmente chamados de anomalias).

Para efetuar o monitoramento de tráfego em redes existem duas técnicas principais: por contadores e por fluxo. No monitoramento utilizando contadores, o administrador precisa frequentemente coletar o valor dos contadores de tráfego do equipamento e fazer uma análise da variação desses dados ao longo de um tempo. Nesse tipo de técnica, a análise do tráfego é feita de maneira agregada, pois os contadores coletados representam vários tipos de tráfego que entram ou saem por uma interface. Apesar do monitoramento baseado em contadores ser bastante eficiente, ele também é escasso de detalhes acerca do tráfego, sendo difícil, por exemplo, reportar quais são as aplicações que mais estão utilizando os recursos de rede.

O segundo tipo de técnica consiste em monitorar individualmente os fluxos de pacotes. De maneira resumida, um fluxo representa um conjunto de pacotes que possuem características similares, por exemplo, pacotes do protocolo HTTP que estão sendo enviados de um *host* A para um *host* B. Por se tratar de uma técnica que oferece um maior nível de granularidade, é possível apontar com uma acurácia adequada quais são os *hosts* e aplicações que estão demandando maior uso de um enlace, por exemplo. A principal desvantagem dessa técnica é a quantidade de dados potencialmente alta que precisa ser tratada, especialmente em equipamentos que encaminham grande número de fluxos. Para as duas técnicas, existem protocolos padronizados e ferramentas disponíveis, as quais auxiliam no processo de coleta e visualização das informações. Na Seção 2.2.2, serão apresentados mais detalhes sobre cada técnica.

Uma técnica alternativa é usada por provedores de Internet, a qual se baseia no cálculo da matriz de tráfego (original do inglês *Traffic Matrix* - TM) da rede. Resumidamente, essa técnica consiste em quantificar o volume de tráfego entre todos os pontos de entrada e saída de uma rede [3], em um dado intervalo de tempo. Utilizando os dados de uma TM, os provedores podem efetuar um melhor provisionamento de capacidade, identificar rapidamente enlaces com problemas, realizar detecção de anomalias de tráfego, entre outras aplicações. Assim como no monitoramento por contadores, uma TM utiliza medições que representam agregações de tráfego entre dois pontos da rede. Dessa forma, seu uso em monitoramento fica restrito, pois essa técnica não oferece um detalhamento adequado sobre o tráfego da rede, por exemplo, não identifica com precisão quais *hosts* são fontes de um ataque.

2.2.1 Amostragem

Uma abordagem simples, para o monitoramento do tráfego em uma rede, consiste em capturar e inspecionar todos os pacotes que trafegam em um enlace. No entanto, existem alguns problemas associados a essa abordagem quando aplicada a enlaces de alta capacidade, por exemplo, 40 Gbps. Nesse tipo de enlace, a quantidade de pacotes recebidos ocorre com taxas elevadas, da ordem de milhões de pacotes por segundo. Esse grande volume de pacotes demanda uma alta carga de processamento, a qual é necessária para efetuar a inspeção dos pacotes em tempo real. Uma alternativa é o armazenamento persistente do tráfego de pacotes, para efetuar o processamento posterior de maneira *offline*. A dificuldade nesse caso é que, dependendo da quantidade e capacidade dos enlaces monitorados, essa abordagem requer uma infraestrutura significativa para armazenamento dos dados. Por último, os ativos de rede, como roteadores e comutadores, são projetados para efetuar operações bem simples nos pacotes, como consultar tabelas de rotas ou configuração de rótulos de VLAN, de forma a garantir o desempenho na transmissão. Além disso, esses equipamentos possuem restrições de processamento e memória e, portanto, não suportam nativamente operações como inspeção detalhada em todos os pacotes.

Essas dificuldades motivaram o desenvolvimento de uma técnica com o objetivo de reduzir a quantidade de dados a ser analizada. A técnica da amostragem, utilizada no campo da estatística, atende esse propósito. Normalmente, essa técnica é utilizada em contextos em que não há viabilidade técnica ou financeira para obter informações sobre toda a população. Basicamente, a amostragem busca selecionar um subconjunto da população que seja capaz de representá-la em termos de alguma característica de interesse. A amostragem aplicada no contexto de monitoramento de redes visa reduzir a quantidade de pacotes capturados sem introduzir distorções em relação às características presentes no tráfego original.

O IETF (*Internet Engineering Task Force*), através do grupo de trabalho em amostragem de pacotes [41], propôs o padrão PSAMP (*Packet SAMPling*). Esse padrão define um conjunto de operações a serem realizadas para a seleção de pacotes [71], especifica quais informações são capturadas a partir das amostras [74], e descreve os protocolos que serão utilizados para exportar as informações para as aplicações [73]. Segundo a RFC 5475 [72], o processo de amostragem pode ser dividido em dois tipos básicos: dependente de conteúdo e independente de contéudo. Na amostragem dependente de conteúdo, o pacote é selecionado de acordo com alguma característica previamente definida do cabeçalho, por exemplo, pacotes com protocolo TCP e porta 25. Na amostragem independente de contéudo, nenhuma parte do pacote é utilizada como critério para seleção da amostra.

Ainda de acordo com o padrão PSAMP, o processo de amostragem pode utilizar os seguintes critérios para seleção de pacotes: sistemático e aleatório. No critério sistemático, o processo de escolha da amostra bem como sua duração são governados por uma função determinística, de forma que as coletas são realizadas de maneira periódica. Nesse tipo de amostragem, existem duas abordagens comuns: por contadores e temporal. Na abordagem por contadores, os pacotes são coletados de acordo com a sequência em que são recebidos, geralmente, na forma de 1 pacote a cada X pacotes em uma janela de medição. Usando a abordagem temporal, as amostras são coletadas de acordo com intervalos fixos de tempo que correspondem ao instante em que os pacotes foram recebidos, por exemplo, 1 pacote a cada 100 ms de tráfego.

Na amostragem aleatória o critério de seleção das amostras é regido por um processo aleatório. Nesse tipo de amostragem, cada pacote coletado representa o resultado de um evento independente. Na amostragem aleatória, podem ser utilizadas estratégias simples ou probabilísticas. Na primeira, n amostras são selecionadas a partir de um conjunto com N elementos, por isso essa técnica também é conhecida como *n-out-of-N*. Na estratégia probabilística, a decisão sobre a coleta ou não de um elemento é baseada em uma probabilidade de seleção previamente definida.

Alguns autores ainda definem outros tipos de amostragem, como o estratificado e o adaptativo. Na amostragem estratificada, os pacotes são divididos em janelas e a amostra é coletada de maneira aleatória em cada uma das janelas [86]. No adaptativo, os parâmetros de coleta são ajustados dinamicamente com o intuito de se obter melhores resultados [50, 83] de acordo com o problema investigado. Hernandez et al. [37] propõem um mecanismo de amostragem adaptativo utilizando uma técnica de predição linear. Hu et al. [39] propõem um método de amostragem adaptativa não-linear com o intuito de reduzir os erros para os fluxos pequenos.

Apesar de trazer benefícios ao monitoramento de pacotes, o uso de amostragem exige planejamento. Caso uma taxa de amostragem seja muito baixa, o conjunto de paco-

tes selecionado pode não capturar características ou eventos de interesse, comprometendo a acurácia do monitoramento. Alguns trabalhos [7, 75] já investigaram e constataram o impacto da amostragem na acurácia do monitoramento.

2.2.2 Redes Clássicas

Com o crescimento do tráfego de pacotes nas redes corporativas e também na Internet, foi necessário adotar uma solução de monitoramento de redes baseadas na pilha TCP/IP. O protocolo SNMP (*Simple Network Management Protocol*) [62] foi criado para ser um padrão para gerenciar remotamente equipamentos de rede de diferentes fabricantes de maneira transparente. O modelo de operação do SNMP é composto por gerentes e agentes, além de protocolos de comunicação para efetuar a troca de mensagens entre essas duas entidades. O gerente tem a função de enviar requisições para os ativos solicitando informações de gerenciamento e também configurar parâmetros operacionais do dispositivo. O agente é um componente que fica acoplado ao equipamento gerenciado, sendo responsável por atender as requisições enviadas pelo gerente, assim como enviar notificações assíncronas de eventos relevantes.

O SNMP define que a comunicação entre o gerente e os agentes seja realizada através da troca de mensagens simples de requisição e resposta, sendo que cada mensagem contém informações de gerenciamento. As informações de gerenciamento são organizadas de maneira hierárquica na forma de variáveis, as quais são agregadas em uma base de informações de gerenciamento (*Management Information Base* - MIB). O gerente SNMP pode realizar operações de leitura e de escrita nessas variáveis. A operação de escrita em uma variável da MIB representa a modificação de um parâmetro de configuração do equipamento, por exemplo, desabilitar uma interface de rede ou adicionar uma regra de controle de acesso.

O padrão SNMP define ainda algumas mensagens síncronas e assíncronas. As mensagens síncronas são utilizadas pelo gerente para requisitar informações ou configurar variáveis da MIB. Seguem alguns exemplos de mensagens síncronas:

- *GetRequest* - requisita o valor de uma variável específica;
- *GetNextRequest* - requisição com a função de descobrir as variáveis disponíveis e seus valores;
- *GetBulkRequest* - versão melhorada da mensagem *GetNextRequest*, na qual múltiplas iterações são realizadas para recuperar o valor de várias variáveis em uma única requisição;
- *SetRequest* - requisita a alteração do valor de uma variável.

As mensagens assíncronas são enviadas do agente para o gerente, sem que o gerente tenha enviado explicitamente uma requisição. A função das mensagens assíncronas

é notificar o gerente que algum evento significativo ocorreu. Normalmente, o endereço de destino das mensagens assíncronas é configurado previamente pelo gerente. Por exemplo, as mensagens *Trap* são enviadas pelo agente em situações de falha nos sub-componentes do equipamento.

Atualmente, o protocolo SNMP é bastante difundido entre os fabricantes de equipamentos de rede e considerado um recurso básico comumente disponível nos dispositivos, dispensando a aquisição de uma licença adicional. Adicionalmente, existem ferramentas que, utilizando o SNMP, automatizam o processo de coleta e visualização de informações, por exemplo: Cacti [8], Zabbix [97], Splunk [84], etc.

A técnica de monitoramento por contadores pode ser realizada utilizando o protocolo SNMP. De fato, as informações de contadores de interfaces são disponibilizadas em variáveis padrões da MIB como *ifInOctets* e *ifOutUcastPkts*, as quais representam respectivamente a quantidade em bytes recebidos e quantidade de pacotes enviados em uma interface. Com a utilização do SNMP, um administrador de rede pode efetuar consultas periódicas aos contadores e, assim, efetuar uma análise desses dados ao longo de intervalos regulares de medição.

Outro método amplamente utilizado para monitorar o tráfego de pacotes na Internet é através da análise dos fluxos. Na literatura, o termo fluxo possui diferentes definições de acordo com o contexto [42, 63, 64]. Nesta dissertação, vamos utilizar a definição proposta na RFC 7011 [43], na qual um fluxo é definido como um conjunto de pacotes ou quadros que passam por um ponto de observação em uma rede durante um certo intervalo de tempo e todos os pacotes pertencentes a um fluxo em particular possuem um conjunto de características em comum. Normalmente, as características em comum mais empregadas são os campos do cabeçalho do pacote IP. A seguir, apresentaremos os aspectos mais relevantes a respeito dos padrões usados para monitoramento de fluxos.

A arquitetura de um sistema típico de monitoramento de fluxos consiste das quatro etapas ilustradas na Figura 2.5. A primeira etapa é a de observação, na qual os pacotes são capturados a partir de um ponto de observação que pode ser uma interface física de um roteador, por exemplo. A segunda etapa consiste na utilização de processos de medição e exportação dos fluxos. Os processos de medição recebem como entrada os cabeçalhos dos pacotes capturados de um ou mais pontos de observação e sua função é efetuar a geração dos registros de fluxos. Os processos de exportação são encarregados de empacotar os registros de fluxos em datagramas e enviar para os dispositivos de coleta. A terceira etapa é a de coleta dos dados que consiste na recepção, no armazenamento e no pré-processamento dos registros de fluxo. A última etapa é a análise dos dados, a qual realiza uma varredura nos registro de fluxos com a finalidade de extrair informações relevantes sobre o tráfego da rede, por exemplo, encontrar os *hosts* com maior consumo de banda, determinar o perfil do tráfego e efetuar correlação de eventos na rede. Adicionalmente,

nessa etapa, o administrador pode utilizar processos manuais para obtenção de relatórios de tráfego ou utilizar processos automatizados para esse fim.

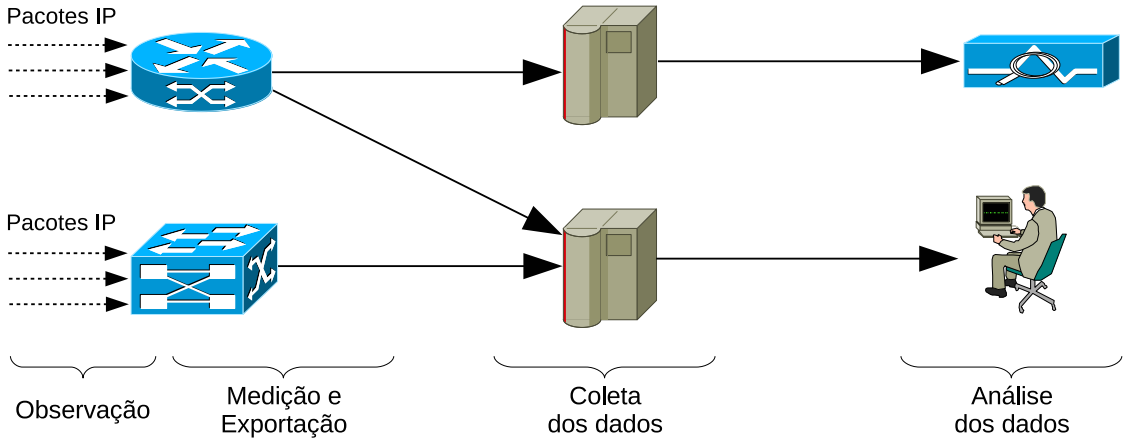


Figura 2.5: *Arquitetura de um sistema de monitoramento de fluxos.*

A primeira solução que acabou sendo adotada como padrão para monitoramento de fluxos foi o NetFlow [13], desenvolvido pela empresa Cisco. Basicamente, o NetFlow coleta estatísticas de pacotes IP que trafegam em roteadores ou comutadores agregando os dados em registros de fluxos que são posteriormente enviados à um dispositivo coletor. Esse dispositivo tem a função de armazenar os registros e efetuar análises de tráfego. Os registros de fluxos são preenchidos com informações coletadas na interface de rede, na qual os pacotes ingressam no equipamento. O padrão define duas maneiras para detectar quando um fluxo termina. Uma dessas maneiras utiliza um contador de inatividade que é zerado assim que novos pacotes são recebidos. No caso de fluxos TCP, a finalização da sessão TCP é utilizada como indicativo de término do fluxo. Outra maneira de detectar o encerramento de um fluxo utiliza intervalos pré-fixados de tempo para fechamento dos registros de fluxo, ainda que existam pacotes desse fluxo sendo recebidos. A partir da análise dos dados recebidos no coletor, é possível identificar algumas características sobre o tráfego da rede, como a lista dos *hosts* que mais consumiram banda ou a distribuição do tamanho dos pacotes.

Apesar do padrão NetFlow ser proprietário da Cisco, a mesma disponibilizou abertamente o formato de dados usado para armazenar os registros de fluxo, permitindo que várias empresas desenvolvedoras de soluções de monitoramento pudessem incorporar em seus produtos o suporte ao padrão.

Para padronizar a forma como as informações de fluxos de pacotes IP são exportadas de roteadores e comutadores, o IETF desenvolveu o padrão IPFIX (*Internet Protocol Flow Information eXport*) [45], definido na RFC 7011 [43]. Inicialmente, a versão 9 do protocolo NetFlow foi escolhida como base para o desenvolvimento e a especificação do padrão IPFIX [38] e, por isso, os dois padrões possuem várias semelhanças. A principal

diferença entre os padrões está no fato do IPFIX permitir que as informações coletadas possam ser customizadas através da utilização de modelos, possibilitando uma flexibilidade na escolha dos campos que serão empregados no monitoramento dos fluxos.

Apesar do monitoramento por fluxos permitir um conjunto detalhado de informações sobre o tráfego da rede, há algumas questões relativas ao uso dessas tecnologias. Inicialmente, dependendo dos níveis de utilização dos enlaces monitorados, a carga de processamento adicional que é associada à geração dos registros de fluxos pode ser bem elevada, ocasionando problemas de desempenho e de acurácia das medições. O emprego de técnicas de amostragem pode contribuir para a resolução dessa questão, uma vez que reduz a quantidade de pacotes que serão inspecionados. No entanto, a amostragem pode levar a uma diminuição na acurácia das medições, sobretudo se taxas de amostragem muito baixas forem utilizadas. Outra questão é a perda de algumas informações relevantes para a caracterização do tráfego, por exemplo, a distribuição dos intervalos de tempo entre os pacotes de um mesmo fluxo, o qual pode ser útil em sistemas de detecção de anomalias.

As soluções clássicas de monitoramento apresentadas se mostram inadequadas em redes de alta capacidade. A primeira abordagem (SNMP) apesar de possuir um bom compromisso entre escalabilidade e tempo de resposta, apresenta um nível de granularidade insuficiente para a maioria das aplicações de gerenciamento de rede. A abordagem através de fluxos consegue monitorar enlaces de alta capacidade utilizando um custo extra de processamento nos dispositivos de rede, além de introduzir um certo grau de erro nas medições devido a utilização de mecanismos de amostragem.

2.2.3 Redes OpenFlow

Os recursos disponíveis no protocolo OpenFlow permitem construir uma arquitetura de monitoramento flexível o suficiente para atender várias tarefas de gerenciamento. Uma vantagem natural é que o controlador tem uma visão global da rede, possibilitando uma coleta simplificada de diferentes tipos de medições em partes distintas da rede. Além disso, na interface programática do controlador existem algumas primitivas que possibilitam a coleta de estatísticas de fluxos nos comutadores. Adicionalmente, uma aplicação no controlador pode efetuar coletas de estatísticas sobre diferentes níveis de agregação de tráfego, apenas alterando as regras instaladas.

Há várias iniciativas empregando OpenFlow para realizar o monitoramento em redes, das quais destacamos algumas relevantes a seguir. Chowdhury et al. [12] propuseram um arcabouço de monitoramento para redes SDN chamado PayLess, o qual oferece algumas vantagens no desenvolvimento de aplicações de monitoramento. O PayLess provê uma visão abstrata da rede e também uma maneira uniforme de requisitar

as estatísticas de tráfego nos ativos. O arcabouço ainda possibilita que as aplicações possam expressar em alto nível os requisitos de monitoramento, enquanto o arcabouço se encarrega de traduzir e aplicar as regras nos equipamentos. Por fim, o PayLess utiliza um algoritmo adaptativo para efetuar as coletas de estatísticas, permitindo um alto grau de precisão nas informações sem incorrer em sobrecarga de processamento nos comutadores.

Em aplicações de engenharia de tráfego o monitoramento precisa prover informações bem acuradas e em tempo real, de forma a permitir um cálculo eficiente dos melhores caminhos. Nesse contexto, a ferramenta OpenNetMon [88] possibilita efetuar medições por fluxo de diferentes métricas como vazão, perda de pacotes e atraso. A ferramenta utiliza o recurso de expiração automática de regras para obter estatísticas de fluxos inativos, através da mensagem *FlowRemoved* que é enviada do comutador para o controlador quando uma regra da tabela de fluxos expira. A OpenNetMon utiliza mais duas formas de coleta de estatísticas, uma através de coletas regulares usando requisições de estatísticas de fluxos e outra através da injeção de pacotes na rede para monitorar o atraso fim-a-fim de um caminho.

É comum assumir que existem recursos suficientes nos comutadores OF para realizar consultas frequentes às estatísticas de fluxo, sem levar em consideração o impacto dessas medições nos ativos e na rede. Moshref et al. [59] investigaram o compromisso entre a acurácia das medições e a utilização de recursos nos ativos de rede, levando em conta o consumo de CPU, memória e banda necessária para o tráfego de controle. Os autores assumem três primitivas básicas para efetuar as medições nos comutadores: 1) contagem de estatísticas de fluxos; 2) contagem baseada em estruturas *hash* e 3) programabilidade. Na primitiva 1), os contadores dos fluxos são transferidos para o controlador, onde é realizada a análise. Na primitiva 2), usando a memória SRAM, o comutador emprega localmente estruturas de dados baseadas em *hash* que extraem estatísticas sobre o tráfego e enviam para o controlador. A primitiva 3) trata da possibilidade de executar códigos simples na CPU do equipamento, permitindo realizar análises de tráfego em uma menor escala de tempo. No entanto, as primitivas 2) e 3) não fazem parte da arquitetura atual de comutadores OF, impossibilitando que esse tipo de solução de medição seja colocado em prática.

As soluções de monitoramento apresentadas possuem alguns problemas que restringem sua utilização em redes OpenFlow reais, devido ao fato de que algumas soluções empregam funcionalidades que não fazem parte da arquitetura atual de comutadores OF, enquanto outras conflitam com o modelo de operação do protocolo OF. Por outro lado, soluções de monitoramento compatíveis com os recursos padrões dos comutadores OF e com o modelo operacional do protocolo permitem sua utilização em redes OF de grande porte. A solução de monitoramento proposta nesta dissertação se baseia nas funcionalidades presentes no protocolo OF, possibilitando sua adoção em redes OF reais [47].

2.3 Agregações Hierárquicas de Fluxos

Parte do esforço para realizar o monitoramento em um enlace de rede envolve efetuar medições periódicas de tráfego de pacotes. As abordagens clássicas para medição de tráfego consistem na contabilização individual de pacotes ou de fluxos de pacotes. No entanto, essas abordagens possuem problemas quando aplicadas a redes de grande capacidade. Nesse tipo de rede, realizar o monitoramento em tempo real de um volume tão grande de tráfego exige um alto custo computacional, uma vez que as taxas envolvidas podem chegar a milhões de pacotes por segundo. Devido a isso, surgiram técnicas que trabalham de maneira eficiente em redes de alta capacidade monitorando apenas os itens mais frequentes. Essa técnica teve sua origem na área de mineração de fluxos de dados [14], onde se estudam formas eficientes para tratar quantidades massivas de dados utilizando algoritmos que possam extrair conhecimento a partir da análise desses dados em tempo real. O problema dos itens mais frequentes se resume em encontrar todos os elementos de um fluxo de dados que aparecem com uma dada frequência acima de um limiar de detecção. Nesse contexto, o termo limiar de detecção se refere a um patamar mínimo no qual todos os fluxos que estejam acima ou igual a esse patamar são classificados como itens frequentes. Na literatura, esse problema também é conhecido como detecção de *heavy hitters* [16], *elephants* [27] ou *alpha flows* [92].

De maneira mais formal, *heavy hitter* (HH) é um fluxo ou um agregado de tráfego que consome uma porção significativa do total de recursos da rede [18]. Assumindo que o total de tráfego, medido em um intervalo regular de tempo, é C e dado um certo limiar de detecção ϕ, um HH é um fluxo ou um agregado de fluxos que consomem pelo menos ϕC do volume total de tráfego. Em um enlace, podem existir no máximo $1/\phi$ HHs simultaneamente. Por exemplo, considerando um limiar $\phi = 5\%$ e um enlace com vazão $C = 1$ *Gbps*, todos os fluxos que estão trafegando a uma taxa de 50 Mbps ou superior são classificados como HHs. Logo, podem haver no máximo 20 *heavy hitters*.

O tráfego de pacotes IP é um exemplo de fluxo de dados que pode ser organizado em uma estrutura hierárquica. De fato, as informações sobre os pacotes dispostas na forma de uma hierarquia são uma representação mais compacta do tráfego como um todo. Exemplos de hierarquias podem ser: agregação dos pacotes em blocos de rede, por protocolos do nível de aplicação, por sistemas autônomos de roteamento, entre outras. A generalização hierárquica do problema de HH é chamado de *hierarchical heavy hitters* [16]. Os *hierarchical heavy hitters* (HHHs) são HHs que estão dipostos em uma dada estrutura hierárquica e que seguem algumas regras simples. Caso o HH esteja localizado no nó mais inferior da hierarquia, esse também é classificado como HHH. Para os níveis superiores, é necessário calcular a contribuição efetiva do nó, sendo essa a diferença

entre o volume do nó[1] corrente e a soma da contribuição efetiva de todos os nós HHHs descendentes na estrutura.

Caso a contribuição efetiva esteja maior ou igual ao limiar de detecção ϕ, então esse nó é classificado como HHH. A notação de agregações hierárquicas de fluxos provê uma forma mais compacta e acurada para monitorar o tráfego em uma rede [21], uma vez que essa notação remove possíveis informações redundantes dos HHs que apenas atingem o limiar por conta de seus sucessores.

Na Figura 2.6, é mostrado um exemplo de estrutura hierárquica composta por uma árvore de prefixos, também conhecida como *Trie*. Cada nó é representado por um volume de tráfego agregado e por uma notação binária que indica um bloco de endereços IP correspondente, por exemplo, o nó $1001*$ representa a sub-rede $144.0.0.0/4$. Considerando $\phi = 5$, pode-se perceber que os nós em preto são HHHs, pois atingem o limiar de detecção após efetuar o desconto dos HHHs descendentes. Além disso, a contribuição efetiva calculada de cada nó HHH está representada na cor azul. Os nós em cinza são apenas HHs por não atingirem o patamar mínimo depois de efetuar o desconto dos HHHs subjacentes. Ao realizar a soma das contribuições efetivas de cada nó HHH o valor resultante é 25, o que equivale a 100% do tráfego total.

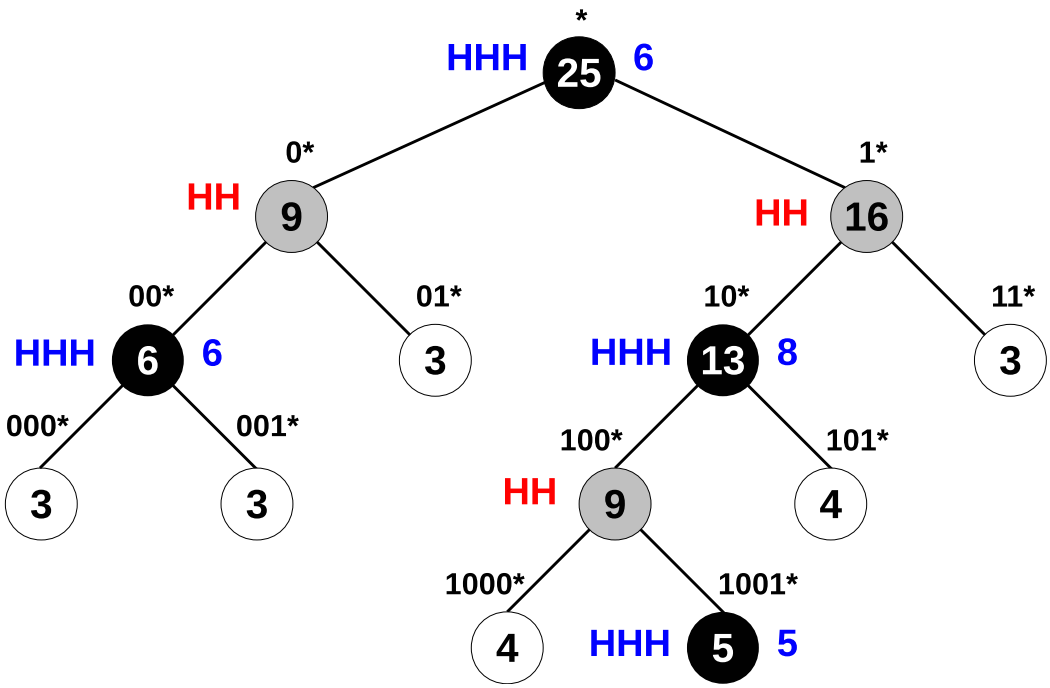


Figura 2.6: *Exemplo de uma estrutura hierárquica contendo HHHs e HHs.*

Vamos considerar outro exemplo, no qual há uma hierarquia parcial composta

[1]Nesse contexto, o volume do nó representa a quantidade de tráfego de pacotes contabilizado em um dado nó da estrutura hierárquica.

pelos seguintes blocos de IP: 10.0.0.0/28 = 13317 pps; 10.0.0.0/30 = 9081 pps; 10.0.0.0/32 = 5010 pps, e os parâmetros $\phi = 5\%$ e $C = 100000$ pps. Nesse exemplo, os blocos 10.0.0.0/28 e 10.0.0.0/32 são HHHs, porém o bloco 10.0.0.0/30 não é HHH porque não alcança o limiar ϕ. Assim, confirmamos que HHH provê uma forma compacta e acurada para representar o tráfego em uma rede.

O problema de detecção de HHHs em tempo real em fluxos de pacotes IP pode ser formulado como segue. Dado um intervalo de medição M, a quantidade total de tráfego C nesse intervalo e um limiar de detecção ϕ, devem ser encontradas todas as agregações hierárquicas de fluxos cujos volumes alcancem no mínimo ϕC após descontar todos os HHHs decendentes.

Apesar dos exemplos anteriores de HHHs considerarem apenas uma informação presente no cabeçalho IP, o conceito de HHH pode ser mais abrangente ao tratar o problema através de múltiplas dimensões, isto é, usando outras informações do cabeçalho do pacote. Por exemplo, um sistema de detecção de HHHs multidimensionais pode considerar os campos de endereço IP de origem, endereço IP de destino, protocolo de transporte, portas de origem e destino da camada de transporte. Embora múltiplas dimensões sejam possíveis, os cenários mais relevantes para análise de tráfego envolvem HHHs unidimensionais e bidimensionais [99]. De maneira genérica, a detecção unidimensional envolve endereços IP de origem ou de destino, enquanto que a detecção bidimensional trabalha com pares de endereços IP de origem e de destino.

Em geral, os algoritmos para detecção de HHHs buscam algumas propriedades comuns, a saber: baixo custo de processamento por item, baixo consumo de memória em comparação com a quantidade de dados tratados e operação online, ou seja, cada item do fluxo de entrada é acessado apenas uma vez enquanto o tempo evolui. Esses algoritmos têm aplicações nas áreas de monitoramento de tráfego de rede, detecção de anomalias e de ataques de negação de serviço distribuído (DDoS) [57]. Nesta dissertação, o foco é a investigação de algoritmos para detecção de HHHs bidimensionais em redes OpenFlow.

2.3.1 Abordagens Clássicas

O problema de detecção de HHHs em redes IP tradicionais já foi amplamente investigado, conforme descreveremos brevemente a seguir. Uma das abordagens propostas consiste em calcular os HHHs com base nas informações do tráfego ao nível de fluxo, empregando os dados coletados por mecanismos como o NetFlow [13]. Conforme comentado na Seção 2.2.2, essas técnicas baseadas em fluxos oferecerem um bom nível de granularidade, mas não mostram escalabilidade em enlaces de alta capacidade. À medida em que a capacidade de transmissão dos enlaces e a quantidade de fluxos aumentam, a manutenção de estatísticas por fluxo nos roteadores e comutadores tende a gerar uma carga

considerável de processamento. Tradicionalmente, são utilizadas técnicas de amostragem para reduzir a quantidade de informação a ser processada, o que traz como efeito colateral a introdução de imprecisão nas medições, conforme verificado por alguns trabalhos da literatura [7, 75].

Embora tenham uso bastante restrito em redes, há alguns trabalhos que propõem algoritmos *offline* para identificação de HHHs. Esses algoritmos assumem que cada item pode ser acessado mais de uma vez e, como consequência, costumam obter resultados com alta acurácia. Por exemplo, Estan et al. [26] propuseram algumas heurísticas para efetuar a detecção *offline* de HHHs multidimensionais. O problema dessa abordagem é que a premissa de operar *online*, respondendo rapidamente a eventos como ataques ou falhas, não é atendida.

Outras abordagens se concentram na identificação dos fluxos elefantes de maneira acurada, enquanto provêem apenas estimativas para os fluxos pequenos. Estan et al. [27] propuseram um algoritmo que realiza a identificação dinâmica de fluxos elefantes, utilizando uma quantidade limitada de memória e apresentando um nível de acurácia superior aos mecanismos que empregam amostragem. Além disso, quanto maior a quantidade de memória usada pelo algoritmo menores são os erros de medição.

É comum na literatura sobre detecção de HHHs propostas que utilizem estruturas de dados probabilísticas [4, 100], também conhecidas como *sketches*. Os *sketches* são estruturas de dados compactas que são usadas em algoritmos de fluxos de dados. Essas estruturas de dados permitem que algumas propriedades chave de um grande volume de dados possam ser aproximadas, utilizando quantidades limitadas de memória. Cormode et al. [16] propuseram o primeiro algoritmo baseado em *sketch* para detecção de HHHs. Esse algoritmo foi projetado para monitorar HHH unidimensionais utilizando um *sketch* diferente para cada nível da hierarquia, possibilitando uma contabilização acurada das agregações hierárquicas.

Em um trabalho posterior, Comorde et al. [17] propuseram algoritmos para o caso de HHHs multidimensionais, direcionados tanto para aplicações *offline*, quanto aplicações *online*. O algoritmo *offline* proposto utiliza uma estrutura de dados chamada *lattice*, a qual armazena os itens de maneira hieráquica. Esse algoritmo realiza vários acessos aos dados de entrada e consegue extrair os prefixos HHHs de maneira exata. Enquanto que os algoritmos *online* propostos obtêm os HHHs de maneira aproximada, mas com uma acurácia alta.

Zhang et al. [99] abordaram a questão de detecção de HHHs propondo algoritmos que empregam algumas técnicas clássicas de classificação de pacotes, uma vez que os autores consideram os dois problemas bastante semelhantes. Os algoritmos propostos utilizam *sketches* adaptativos que mapeam pacotes em conjuntos de prefixos, formando uma estrutura hierárquica. Um desses algoritmos faz parte da solução de detecção pro-

posta nesta dissertação e seus detalhes de funcionamento serão apresentados na Seção 3.3.

Mitzenmacher et al. [57] propuseram uma solução que utiliza múltiplas instâncias do algoritmo *Space Saving* (SS) [56] para detectar HHHs bidimensionais em redes IP convencionais. Nesse trabalho, os autores também utilizaram a *lattice* como estrutura de dados hierárquica. Em cada nó dessa estrutura é executada uma instância do algoritmo SS. Para cada novo item do fluxo de pacotes, o algoritmo calcula todas as suas generalizações e as insere separadamente em cada nó da estrutura. A quantidade de memória utilizada pelo algoritmo não depende do número de itens do fluxo de pacotes, mas do número de contadores mantidos em cada instância do algoritmo SS. Os autores ainda apresentaram algumas extensões ao algoritmo, como implementações distribuídas, paralelas e também com a utilização de tabelas TCAM.

Os trabalhos propostos para redes IP tradicionais possuem algumas deficiências em comum. Como se baseiam no modelo em que controle e dados estão no mesmo dispositivo, a identificação de HHHs deve ser feita no roteador ou comutador. Assim, surgem questões como desempenho do equipamento, escalabilidade, custo da solução, dificuldades para desenvolvimento, manutenção e evolução de soluções embarcadas. Abordagens alternativas, como espelhamento de tráfego para processamento externo são difíceis de serem aplicadas a redes de alta capacidade devido ao grande volume do tráfego agregado. Redes OpenFlow oferecem novas oportunidades, porém novos desafios.

Em comutadores OpenFlow não há lógica de controle interna e também não é viável enviar informações de cada pacote para o controlador. Por outro lado, soluções baseadas em dispositivos OpenFlow podem fazer uso de regras de monitoramento com diferentes níveis de granularidade, contabilizando o tráfego em estruturas flexíveis internamente no controlador. A solução para detecção de HHHs proposta nesta dissertação se baseia nesse princípio.

2.3.2 Abordagens Definidas por Software

A seguir apresentamos os principais trabalhos que utilizam abordagens OpenFlow e SDN para detecção de HHHs ou outras tarefas de monitoramento de redes.

Curtis et al. [20] apresentaram um estudo sobre o impacto do provisionamento de medições de baixa granularidade e visibilidade global nas redes OpenFlow. Os autores argumentam que o atual modelo do OpenFlow envolve o tratamento de muitas regras do tipo *microflow*[2], o que ocasiona uma carga extra de processamento tanto do lado do controlador quanto no comutador. Para evitar esse processamento excessivo, é proposta

[2]As regras do tipo *microflow* são instaladas quando são especificados todos os campos da regra OpenFlow, sem o uso de campos coringa.

uma alteração no modelo operacional do OpenFlow. O objetivo é repassar o controle da maioria dos fluxos para os comutadores, enquanto que a atuação do controlador é restrita apenas aos fluxos mais significativos. Para atingir esse objetivo, os autores propõem que uma parte significativa do controle sobre os fluxos seja devolvida ao comutador que sugerem algumas melhorias na coleta de estatísticas do OpenFlow. Essas melhorias envolvem a utilização de amostragem, gatilhos e contadores aproximados. Apesar das vantagens alegadas pelos autores, há grande dificuldade de adoção da proposta. A principal restrição é o conflito com o modelo adotado para o padrão OpenFlow. Além disso, a fronteira exata que separa as funcionalidades do comutador e do controlador tende a levar a um extenso debate.

Mogul e Congdon [58] propuseram a utilização de contadores definidos por software para permitir um processamento mais flexível no tratamento de estatísticas de tráfego em redes OpenFlow. A proposta é baseada na substituição dos contadores tradicionais (armazenados em chips ASIC) por um pequeno *buffer* que guarda as informações a respeito dos fluxos recentes. Quando o *buffer* está cheio, as informações contidas nele são encaminhadas à CPU do equipamento que armazena os contadores na memória DRAM. Essa abordagem traz bastante flexibilidade na forma de coleta dos contadores, mas por outro lado apresenta sérias restrições quanto a sua implantação em redes reais, pelo fato de utilizar funcionalidades não disponíveis na atual arquitetura de comutadores OpenFlow.

Yu et al. [96] propuseram uma nova arquitetura de medição baseada em software chamada OpenSketch, no qual o plano de dados de medição é separado do plano de controle. No plano de dados, o sistema utiliza uma sequência de três estágios composta por funções de *hashing*, filtragem e contagem. As funções de *hashing* são usadas para reduzir a quantidade de dados a ser medida, enquanto que as funções de filtragem (instaladas na TCAM) efetuam a seleção dos fluxos. O estágio final é composto por funções de contagem armazenadas na SRAM e responsáveis por agregar estatísticas de tráfego. Os contadores podem ser associados a diferentes entidades no sistema, como regras *microflow*, coringa (*wildcard*) ou valor *hash*, permitindo assim uma maior flexibilidade na coleta dos dados. No plano de controle, os algoritmos de medição podem ser implementados usando uma biblioteca que aloca dinâmicamente os recursos necessários no plano de dados. Apesar do OpenSketch possibilitar a coleta de estatísticas de tráfego de maneira flexível, sua arquitetura assume a existência de recursos inexistentes na maior parte dos comutadores disponíveis atualmente no mercado.

Todos os trabalhos anteriores abordam temas relevantes para o monitoramento que podem ser úteis para a detecção de HHHs. No entanto, nenhum deles apresentam uma solução para esse problema no contexto de redes OpenFlow. Jose et al. [49] foram os primeiros a propor uma solução para o problema de detecção de HHHs levando em conta as peculiaridades das redes OpenFlow. A solução apresentada se baseia na identificação

de HHHs unidimensionais através da instalação e da manutenção de regras de monitoramento nos comutadores OpenFlow. Uma aplicação no controlador efetua periodicamente a leitura dos contadores associados às regras de monitoramento. Os valores coletados dos contadores são armazenados no controlador em uma estrutura hierárquica, facilitando o cálculo dos HHHs. Em intervalos regulares de medição, o controlador ajusta a granularidade das regras para permitir a especialização de partes da estrutura, a fim de detectar HHHs com prefixos mais específicos. Apesar do mecanismo proposto possuir vantagens em termos de acurácia e baixa carga de processamento, a técnica empregada na expansão da estrutura necessita de um tempo consideravelmente longo para encontrar prefixos mais específicos, sendo necessários vários intervalos de medição. Esse comportamento torna a solução inadequada em cenários em que o comportamento do tráfego pode se alterar rapidamente. Em enlaces de *backbone* e até mesmo enlaces de acesso de capacidade mais alta, é comum flutuações rápidas no tráfego, sobretudo em condições anômalas como falhas ou ataques. Nesse contexto, a detecção rápida de mudanças no conjunto de HHHs é fundamental para alertar o operador da rede em tempo de tomar contramedidas.

Na tabela 2.3 é mostrada uma comparação com as características mais relevantes sobre os trabalhos anteriores em detecção de agregações hierárquicas de fluxos.

Trabalho	Solução	Abordagem	Dimensão	Granulosidade
Cormode et al. [16]	*online*	convencional	unidimensional	bit
Cormode et al. [17]	*online*, *offline*	convencional	multidimensional	bit
Cruz et al. [21, 22]	*online*	definida por software	bidimensional	bit
Estan et al. [27]	*online*	convencional	–	byte
Jose et al. [49]	*online*	definida por software	unidimensional	bit
Mitzenmacher et al. [57]	*online*	convencional	bidimensional	byte, bit
Zhang et al. [99]	*online*	convencional	unidimensional, bidimensional	bit

Tabela 2.3: *Comparação dos trabalhos anteriores.*

2.4 Conclusão

Nesse capítulo foram apresentados os fundamentos teóricos mais relevantes ao contexto da dissertação. Foram introduzidos os conceitos das Redes Definidas por Software, mostrando suas vantagens em comparação com a abordagem convencional,

além disso foram apresentados os principais aspectos do protocolo OpenFlow, que é a proposta mais adotada para redes SDN. Foram destacadas também as soluções de monitoramento de tráfego utilizadas nas redes convencionais, bem como suas vantagens e desvantagens. Nesse contexto foi apresentada a técnica de monitoramento dos itens mais frequentes que possibilita o monitoramento em redes de alta capacidade, em especial a técnica de monitoramento de agregações hieráquicas de fluxos, que é o foco desse trabalho. E por fim, foram apresentados os principais trabalhos relacionados focados em monitoramento de redes convencionais e de redes OpenFlow, e também alguns trabalhos anteriores sobre detecção de agregações hieráquicas.

No próximo capítulo serão apresentados os detalhes da solução de monitoramento proposta nesse trabalho destacando suas vantagens, em relação a trabalhos anteriores, como baixo tempo de resposta, maior granulosidade na identificação dos fluxos e aderência completa ao padrão OpenFlow.

CAPÍTULO 3

Proposta

Neste capítulo, são apresentados os detalhes de funcionamento das partes que compõem a solução proposta. Inicialmente na Seção 3.1 são apresentados alguns aspectos gerais acerca da nossa solução e sobre o monitoramento de agregações hierárquicas. Na Seção 3.2 é mostrado o ciclo de operação da aplicação OpenFlow, a estrutura hierárquica utilizada para contabilização do tráfego e os procedimentos empregados para fazer a chamada ao Coletor. Por fim, na Seção 3.3, apresentamos o algoritmo utilizado no componente Coletor e seus detalhes de funcionamento. Discutimos ainda, alguns aspectos particulares desse algoritmo que restrigem seu uso como um componente da solução proposta e as modificações implementadas.

3.1 Estrutura Geral

A solução apresentada neste trabalho tem o objetivo de encontrar os HHHs bidimensionais que estejam presentes em um tráfego de pacotes que passa em um comutador OpenFlow. Assumindo que o tráfego tem predominância de pacotes IPv4, o problema de encontrar os HHHs consiste em identificar, de maneira dinâmica, todas as chaves que possuam um volume de tráfego associado que seja igual ou superior a um determinado limiar de detecção ϕ, em um dado intervalo de medição M. No contexto deste trabalho, uma chave é composta por dois campos do cabeçalho do pacote, endereço IP de origem e endereço IP de destino, sendo que cada campo de uma chave é composto por um valor que pode estar entre 0 e 2^{32}. Realizar o monitoramento de todas as chaves possíveis, requer efetuar o produto cruzado entre os campos da chave, o que constitui uma tarefa bem complexa para um comutador OF, devido as suas restrições de processamento e memória.

Nossa solução para identificação de HHHs é chamada de ODHIn – *Online Detection of Hierarchical heavy hitters improved by samples of In-depth inspection* – e não requer nenhuma modificação no *hardware* do comutador, nem no modelo de operação do OpenFlow. O ODHIn é composto por dois componentes principais, uma aplicação OF e um Coletor, conforme ilustrado na Figura 3.1. O primeiro componente, implementado

como uma aplicação no controlador OF, tem a função de efetuar a leitura dos contadores associados às regras instaladas na tabela de fluxos do comutador OF e ajustar essas regras para adequar o nível de granulosidade do monitoramento.

O componente Coletor foi projetado como um dispositivo dedicado conectado ao comutador OpenFlow e tem a função de ajudar a aplicação OF a identificar os HHHs mais rapidamente. Para isso, a aplicação OF instala uma regra específica no comutador que envia uma cópia do tráfego de pacotes para o Coletor. O Coletor ao receber esse tráfego, realiza uma análise *online* nesse conjunto de pacotes para identificar as agregações hierárquicas correspondentes. Ao final dessa análise o Coletor envia o conjunto de HHHs identificados para a aplicação OF, a qual utiliza esse conjunto para efetuar a atualização das regras no comutador. Nos comutadores atuais, nos quais é comum estarem disponíveis múltiplas interfaces de 1 ou 10 Gbps, até uma amostra de 1 segundo do tráfego pode conter milhões de pacotes. Logo, há um custo computacional potencialmente alto para analisar essa quantidade de informação, dificultando severamente o tratamento em tempo real dos dados. Por isso, o Coletor emprega amostragem com o intuito de reduzir o custo computacional e o tempo de resposta, além de aumentar a escalabilidade da solução. Nas próximas seções, apresentaremos maiores detalhes sobre a aplicação OpenFlow e também sobre o Coletor.

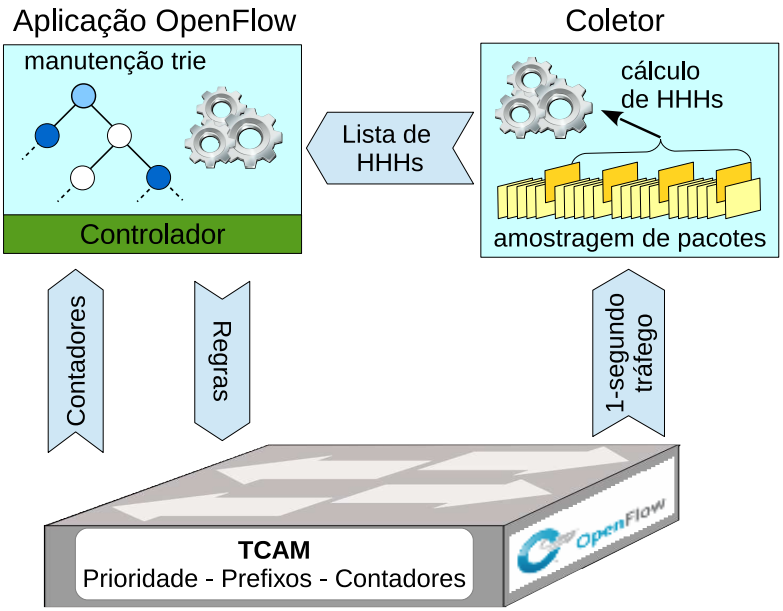


Figura 3.1: *Componentes do ODHIn.*

3.2 Aplicação OpenFlow

A aplicação OF, utilizando um conjunto de abstrações no controlador, envia periodicamente requisições aos comutadores para coletar estatísticas dos contadores asso-

ciados às regras de fluxo instaladas. O ciclo de operação da aplicação OF consiste em: 1) efetuar a leitura dos contadores associados às regras; 2) atualizar a estrutura de dados no controlador; 3) realizar a manutenção do conjunto de regras de monitoramento no comutador; 4) efetuar a chamada ao Coletor caso seja necessário e calcular os prefixos HHHs correspondentes. Todas essas operações ocorrem dentro de um intervalo de medição M. Revisitando o funcionamento convencional de um comutador OpenFlow, para cada pacote que ingressa, o dispositivo realiza a combinação entre as regras instaladas e os cabeçalhos do pacote. Quando o equipamento encontra uma regra correspondente, os contadores da regra são incrementados e o campo de ação da regra é executado. Nossa aplicação OF coleta regularmente o contador de bytes das regras, obtendo uma informação acurada sobre o tráfego da rede. Naturalmente, o nível de granulosidade da regra é um parâmetro crítico para efetivamente usufruir dessa acurácia.

A aplicação OF utiliza uma árvore de prefixos, também conhecida como árvore *Trie*, como estrutura de dados hieráquica para contabilizar o tráfego que atravessa o comutador. Cada nó da árvore armazena os contadores referente a um par específico de prefixos, composto por endereços IP de origem e de destino. Dessa forma, o tráfego de pacotes IP é contabilizado de maneira bidimensional, sendo que a árvore mantém as informações com uma granulosidade em nível de bit. Os pares de prefixos presentes em cada nó da árvore são mapeados no comutador na forma de regras de monitoramento. As regras de monitoramento são instaladas no comutador com a utilização de regras curinga (*wildcard*), um recurso presente nas tabelas TCAM dos equipamentos. Por exemplo, o nó da árvore representado pelo par de prefixos, em notação binária, $1101*, 0100*$ é mapeado para a regra *wildcard*, em notação decimal, $208.0.0.0/4 - 64.0.0.0/4$. Para realizar a atualização adequada de contadores, o mapeamento dos nós da árvore em regras OF é efetuado utilizando o recurso de priorização de regras. Assim, regras com prefixos mais longos recebem uma prioridade maior. A Figura 3.2 ilustra uma parte de uma árvore criada por nossa aplicação OF.

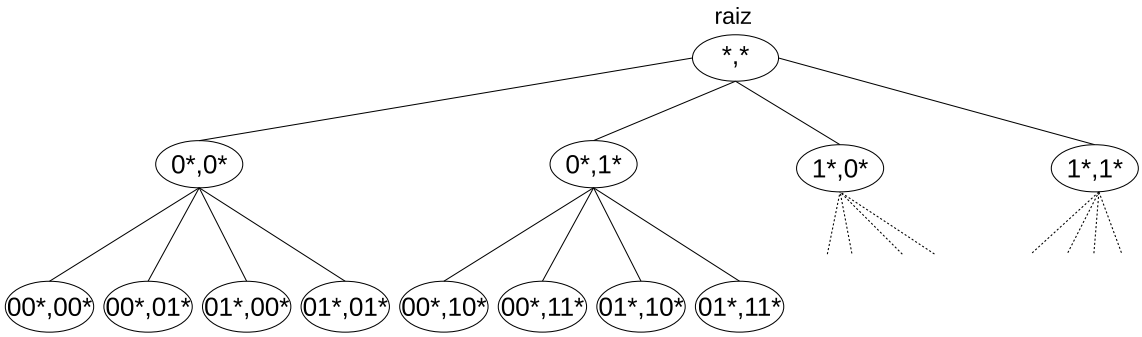


Figura 3.2: *Árvore trie parcialmente construída.*

Inicialmente, a aplicação OF cria a árvore de prefixos efetuando a expansão

dos pares de prefixos a partir da raíz. A quantidade de níveis da árvore *trie* inicial é definida como uma função do limiar de detecção ϕ, mas apresentaremos mais detalhes posteriormente. Cada nó da árvore possui no máximo quatro nós filhos, uma vez que individualmente os prefixos de origem e destino podem ser expandidos um bit por vez. Após a construção inicial da árvore no controlador, a aplicação OF instala as regras de monitoramento no comutador utilizando apenas os nós pertencentes ao último nível da hierarquia, o que corresponde aos nós folhas da árvore. Esse procedimento é realizado para diminuir a quantidade de entradas utilizadas na tabela TCAM, uma vez que esse recurso é limitado nos atuais comutadores OF.

A aplicação OF utiliza uma hierárquia pré-construída, cuja altura é definida logo na inicialização da solução. Essa altura é definida como uma função do número máximo de HHHs, cujo valor é $1/\phi$. Assim, inicialmente, a estrutura da árvore é perfeitamente balanceada e possui uma altura equivalente a $\lceil log_4(1/\phi) \rceil$. Por exemplo, considerando $\phi = 5\%$ e $C = 1Gbps$, a aplicação OF constrói uma árvore inicial com altura igual a 3.

Para assegurar que as informações presentes na árvore estejam acuradas em relação às estatísticas da tabela TCAM, o intervalo de medição M é dividido em *slots* de 1 segundo. Ao final de cada *slot*, a aplicação OF efetua a leitura dos contadores na tabela do comutador e atualiza o volume de tráfego associado aos nós da árvore. Por exemplo, se considerarmos $M = 5s$, então os contadores da tabela TCAM são lidos cinco vezes dentro de um intervalo de medição.

Quando um nó da árvore alcança uma proporção ϕ em relação ao total de tráfego, a aplicação OF efetua uma chamada ao Coletor e instala uma regra temporária no comutador que realiza o espelhamento de tráfego. A regra de espelhamento utiliza o recurso de expiração automática por tempo do protocolo OpenFlow, o *hard-timeout*. Ou seja após um tempo pré-configurado essa regra é removida da tabela do comutador, independente de haver ou não tráfego que esteja combinando com essa regra. Por questões de escalabilidade, a regra de espelhamento é instalada com um tempo de expiração fixado em 1 segundo. Além disso, a instalação da regra de espelhamento faz uso do recurso de múltiplas tabelas de regras do protocolo OpenFlow. Dessa forma, a regra de espelhamento é instalada na tabela 0, enquanto que as demais regras de monitoramento são instaladas na tabela 1 do comutador. Esse procedimento é realizado para evitar uma desabilitação temporária das regras de monitoramento na aplicação OF, uma vez que o espelhamento utiliza uma regra mais abrangente e com maior prioridade. Caso os dois tipos de regras fossem instaladas na mesma tabela, os contadores associados às regras de monitoramento não seriam atualizados enquanto a regra de espelhamento estivesse ativa. A regra da tabela 0 possui duas ações associadas, a primeira encaminha os pacotes para serem processados pela tabela 1 (usando a instrução *Goto*), enquanto a segunda realiza efetivamente o espelhamento do tráfego, ou seja, uma cópia de cada pacote é enviada para a porta onde

o Coletor está conectado.

O Coletor, ao ser invocado pela aplicação OF, realiza uma inspeção no conjunto de pacotes que foi recebido. Ao final de sua execução, o Coletor envia ao controlador a lista de HHHs encontrados no conjunto de pacotes recebidos. A aplicação OF, ao receber essa lista faz a atualização na estrutura da árvore, realizando expansão de nós ou remoção de nós monitorados. O processo de remoção de nós da árvore ocorre quando é detectado que algum nó monitorado não recebe tráfego, ou seja, os contadores correspondentes a regra OpenFlow instalada no comutador não são incrementados. Essa situação pode ocorrer em condições normais da rede, por exemplo, quando há mudança no perfil de tráfego. Para manter a sincronização das informações entre a árvore *trie* do controlador e a tabela TCAM do comutador OF, as operações de expansão e remoção de nós da árvore são replicadas para o comutador, ou seja, as regras OF correspondentes são instaladas ou removidas da tabela de fluxos. No término do intervalo de medição M, a estrutura da árvore é examinada para identificar os respectivos prefixos HHHs do intervalo e a lista final é reportada ao administrador.

No contexto desta dissertação, foram desenvolvidas duas versões da aplicação OF: uma versão na forma de um simulador [22], e uma versão para ambiente real [21]. A primeira versão foi desenvolvida com a finalidade de realizar uma prova de conceito e validação do algoritmo. A segunda versão foi implementada na forma de uma aplicação para o controlador POX [77], sendo plenamente funcional para uso em redes OpenFlow reais. Uma representação em alto nível da aplicação OF é apresentada no Algoritmo 3.1.

As principais diferenças da versão simulador em relação a versão *online* são: 1) forma de obtenção das estatísticas de tráfego (linha 7), na versão simulador a leitura dos contadores é feita diretamente a partir de um arquivo de *trace* em disco, enquanto na versão *online* os contadores são obtidos através de chamadas nativas do OpenFlow; 2) na versão simulador, apenas as operações de manutenção da árvore *trie* são realizadas, enquanto as demais operações de instalação e manutenção de regras *wildcard* não são realizadas (linhas 2, 12 e 15); 3) na versão simulador, o Coletor é invocado através de uma chamada ao processo que está executando no mesmo *host* que o simulador (linha 13).

3.3 Coletor

O componente Coletor é chamado pelo controlador quando um prefixo monitorado da árvore atinge um dado limiar de detecção ϕ. Dado um fluxo de pacotes e um limiar de detecção, o Coletor é um algoritmo aproximado que processa de maneira *online* esse fluxo e retorna uma lista com os prefixos HHHs bidimensionais com seus respectivos volumes de tráfego. Como a solução do ODHIn é projetada para tratar grandes agregados

Algoritmo 3.1: Aplicação OpenFlow

```
entrada: ϕ: limiar de detecção de HHH
         intervalos: conjunto de intervalos de medição
         coletor: referência do processo do Coletor
saída  : listaHHH: lista de HHHs de cada intervalo de medição
1  arvore ← criaArvoreInicial();
2  instalaRegrasCuringa(arvore);
3  para cada M em intervalos faça
4      volumeTotal ← 0;
5      slots ← divideIntervalosEmSlots(M);
6      para cada slot em slots faça
7          contadores ← lerContadoresTCAM();
8          volumeSlot ← calculaVolumeSlot(contadores);
9          volumeTotal ← volumeTotal + volumeSlot;
10         arvore ← atualizaContadores(arvore, contadores);
11         se temHH(arvore, ϕ, volumeTotal) então
12             instalaRegraEspelhamento();
13             listaHHH ← coletor.calculaHHHs(ϕ);
14             arvore ← mesclaHHHs(arvore, listaHHH);
15             instalaRegrasCuringa(arvore);
16         fim
17     fim
18     listaHHH ← extraiHHHs(arvore, ϕ, volumeTotal);
19     reporta listaHHH;
20 fim
```

de tráfego, o Coletor emprega a técnica de amostragem com a finalidade de manter um baixo tempo de resposta e evitar a utilização excessiva de processamento e memória. É utilizado o critério sistemático para seleção dos pacotes em conjunto com a amostragem temporal, na qual as amostras são coletadas em intervalos fixos de tempo, por exemplo, 1 pacote a cada 50 μs.

A implementação do Coletor, utilizada no ODHIn, é baseado no algoritmo bidimensional proposto por Zhang et al. [99]. Esse algoritmo foi escolhido porque trata eficientemente fluxos de pacotes IP de maneira *online* e também por utilizar uma hierarquia com granulosidade em nível de bits, i.e. a mesma empregada na aplicação OF. O algoritmo de Zhang et al. utiliza três estruturas de dados: duas *tries* $t1$ e $t2$ que mantêm a contabilização dos endereços IP de origem e de destino respectivamente, e uma matriz H com dimensões WxW. O valor de W representa o comprimento máximo dos prefixos em cada dimensão, ou seja, para o protocolo IPv4 o valor de W é 32. Cada célula da matriz H contém uma tabela *hash* que é utilizada para manter os prefixos bidimensionais, assim como seus respectivos volumes.

O Algoritmo 3.2 apresenta o pseudo-código do Coletor. Para cada elemento e

proveniente de um fluxo de pacotes, o algoritmo encontra uma tupla $< p_1, p_2 >$ (linhas 5 e 6) que corresponde ao prefixo mais longo que combina com os endereços IP de origem e de destino de e nas *tries* $t1$ e $t2$ respectivamente. Em seguida, o volume é atualizado nas *tries* com o valor correspondente ao elemento e, efetuando a expansão das árvores conforme necessário. A implementação de cada *trie* utiliza a estrutura de dados de uma árvore binária, na qual um novo nó é criado com base em um limiar do algoritmo, chamado T_{split}. O limiar T_{split} é definido como uma função entre o volume total (SUM) de tráfego, o erro máximo tolerado pelo algoritmo (ε) e a altura máxima da árvore (W):

$$T_{split} = \frac{\varepsilon \cdot SUM}{W}. \tag{3-1}$$

De acordo com a Equação 3-1, quando um dado nó da *trie* atinge o limiar T_{split}, esse se torna um nó interno e um novo nó folha é criado. Nessa etapa do algoritmo, as *tries* reportam o prefixo mais longo (p_1 e p_2) correspondente ao elemento e do fluxo de pacotes. A expansão das *tries* unidimensionais não depende do limiar de detecção de HHHs ϕ. Dessa forma, o algoritmo apresenta o mesmo comportamento para diferentes valores de ϕ, desde que seja mantido o mesmo fluxo de pacotes. É importante ressaltar que o critério de expansão da *trie* utilizada no algoritmo proposto por Zhang et al. difere daquele usado no algoritmo da aplicação OF. Nesse último, o critério de expansão das *tries* leva em consideração o limiar ϕ.

Utilizando os comprimentos l_1 e l_2 correspondentes aos prefixos p_1 e p_2, o algoritmo compõe uma chave que é a concatenação dos prefixos p_1 e p_2 (linha 10). Com a chave resultante, uma nova entrada é criada, caso não exista, na tabela *hash* correspondente a posição $H[l_1][l_2]$, e seu volume é inicializado com o valor de e (linha 11). Caso a chave resultante já exista em uma tabela *hash* da matriz H, o volume correspondente à entrada é incrementado com o valor de e. Em resumo, os elementos da matriz H são populados a partir dos prefixos retornados (p_1 e p_2) pelas *tries* unidimensionais t_1 e t_2.

Após processar todo o fluxo de pacotes, o algoritmo efetua a reconstrução de volume dos elementos intermediários da matriz H (linha 13). Para isso, o algoritmo incrementa o volume de cada nó nos seus antecessores. Esse processo é realizado de maneira decrescente na estrutura, ou seja, do final da hierarquia para o início. Finalmente, o algoritmo processa toda a matriz H para extrair os HHHs (linhas 14 e 15), enviando a lista com os prefixos, juntamente com seus volumes, para a aplicação OF.

O algoritmo utilizado no Coletor [99] não é exato, isso porque as *tries* ($t1$ e $t2$) que contém informações unidimensionais apenas são expandidas quando o volume associado atinge um certo limiar. Outro detalhe, é que esse algoritmo utiliza um valor estimado para o volume associado aos nós das estruturas de dados. Isso é feito para compensar um certo percentual de pacotes não contabilizados, que pode ocorrer em

Algoritmo 3.2: Coletor

entrada: ϕ: limiar de detecção de HHH
S: conjunto de pacotes amostrados
saída : $listaHHH$: lista de HHHs resultante das amostras

```
1  t1 ← NULO;
2  t2 ← NULO;
3  Inicializa cada elemento w_{i,j} em H com uma tabela hash m;
4  para cada e em S faça
5      p1 ← atualizaArvore(t1, e, ip_origem);
6      p2 ← atualizaArvore(t2, e, ip_destino);
7      l1 ← comprimento(p1);
8      l2 ← comprimento(p2);
9      m ← H[l1][l2];
10     chave ← p1 + p2;
11     atualizaHash(m, chave, e);
12 fim
13 reconstroiVolumeMatriz(H);
14 volumeTotal ← calculaVolumeTotal(H) ;
15 listaHHH ← extraiHHH(H, φ, volumeTotal);
16 reporta listaHHH;
```

situações de tráfego intenso, enquanto o algoritmo estava realizando operações como expansão e atualização de volume dos nós. Para isso, o algoritmo calcula um valor aproximado para o tráfego de pacotes que é perdido durante essas operações.

A versão original do algoritmo escolhido [99] para o Coletor apresenta dois aspectos que impedem o seu uso adequado como parte do ODHIn: 1) dependendo da parametrização utilizada, o algoritmo é incapaz de detectar os prefixos HHHs; 2) o algoritmo utiliza uma definição diferente de HHH. Portanto, realizamos algumas alterações no algoritmo para tratar esses aspectos e adequá-lo à nossa demanda. A seguir, descrevemos como as alterações foram realizadas, partindo de uma descrição detalhada dos aspectos listados.

O aspecto 1) se manifesta quando o algoritmo recebe uma quantidade pequena de pacotes como consequência de uma baixa utilização da infraestrutura, comum em determinados períodos (ex.: de madrugada) em redes de acesso. Nesse tipo de cenário, o volume total de tráfego SUM é baixo e, portanto, resulta em um baixo valor para o limiar T_{split}. Como consequência, o algoritmo efetua a expansão das *tries* unidimensionais muito rapidamente pois, geralmente, o volume associado ao elemento e está acima do valor de T_{split}. Assim, os prefixos p_1 e p_2 passam a alcançar, na maior parte dos casos, o comprimento máximo da estrutura, ou seja, W. Dessa forma, as tabelas *hash*

intermediárias[1] da matriz H não são populadas e, como consequência, o algoritmo não consegue identificar os HHHs com tamanho de prefixos inferiores a W. Para ilustrar esse tipo de cenário, segue um exemplo: o algoritmo recebe 500 pacotes com tamanho de 1500 bytes e usa $\varepsilon = 0,001$, logo, $T_{split} = 23,4$, dado que W tem seu comprimento máximo de 32. Portanto, todos os pacotes recebidos pelo algoritmo tem volume acima do valor de T_{split}. Nessa situação, de acordo com o algoritmo original, a expansão das *tries* unidimensionais ocorre de maneira rápida, bastando processar apenas 1 pacote para que as árvores expandam completamente.

Para lidar com o aspecto 1), introduzimos um parâmetro k que controla a expansão das *tries* unidimensionais. Esse parâmetro representa a quantidade de níveis que o algoritmo deve ramificar, ao processar cada item e do fluxo de pacotes. Caso o parâmetro k tenha um valor alto, as *tries* são expandidas mais rapidamente, caso contrário a estrutura cresce lentamente. O benefício dessa modificação é que o algoritmo consegue popular as tabelas *hash* intermediárias da matriz H, mesmo que o Coletor receba uma quantidade pequena de pacotes. Na etapa de reconstrução dos volumes da matriz, os contadores correspondentes aos elementos das tabelas intermediárias são corretamente incrementados.

Com relação ao aspecto 2), o algoritmo original [99] reporta todos os prefixos que estão acima do limiar de detecção ϕ, ou seja, os prefixos resultantes incluem HHs e não apenas os HHHs. Essa característica dificulta a fusão do conjunto de prefixos HHHs retornado pelo algoritmo com a estrutura de dados mantida na aplicação OF, a qual emprega a definição estrita de HHH. Zhang et al. [99] utilizam essa definição diferente para facilitar a aplicação do algoritmo em cenários onde é relevante a detecção de mudanças no conjunto de HHHs.

A abordagem para tratar o aspecto 2) consiste em modificar o cálculo dos HHHs do algoritmo para utilizar a mesma definição que foi empregada na aplicação OF, ou seja, os nós HHHs identificados que possuem uma contribuição efetiva, após descontar o volume dos prefixos HHHs descendentes, acima do limiar de detecção ϕ. Assim, o algoritmo consegue identificar adequadamente apenas os HHHs correspondentes ao fluxo de pacotes recebido.

3.4 Conclusão

Neste capítulo, apresentamos em detalhes o ODHIn, nossa solução para identificação *online* de agregações hierárquicas bidimensionais de fluxos projetada para operar

[1] Nesse contexto, o termo tabelas *hash* intermediárias está sendo utilizado para denotar todos os elementos $H[l_1][l_2]$ tal que $1 \leq l_1, l_2 < W$.

em redes OpenFlow. A solução é composta por dois componentes, uma aplicação OF e um sistema chamado Coletor. O primeiro componente interage com os comutadores de rede, a fim de obter estatísticas associadas as regras OF da tabela TCAM. Para contabilizar adequadamente o tráfego de pacotes esse componente utiliza uma estrutura de dados bidimensional, capaz de armazenar as informações sobre as agregações de tráfego de maneira hierárquica. O segundo componente é capaz de lidar com amostras de tráfego e tem a função de identificar rapidamente os prefixos HHHs presentes nesse conjunto de amostras, reportando para a aplicação OF.

No próximo capítulo, apresentaremos a avaliação de desempenho da solução proposta. Apresentaremos algumas avaliações de acurácia em comparação com uma proposta anterior da literatura utilizando métricas clássicas, além disso será introduzida uma nova métrica para capturar de maneira mais efetiva o comportamento das soluções. Também mostraremos alguns experimentos para avaliar as melhorias trazidas pelas modificações no algoritmo do Coletor, e o impacto da utilização de amostragem sobre esse componente.

CAPÍTULO 4

Avaliação

Neste capítulo, apresentamos a avaliação experimental da solucão proposta. Inicialmente, na Seção 4.1, descrevemos os ambientes em que os experimentos foram realizados. Na Seção 4.2, introduzimos as métricas utilizadas na avaliação, incluindo uma uma nova métrica baseada em similaridade, a qual propusemos com a finalidade de melhor avaliar soluções com o mesmo propósito do ODHIn. Em seguida, na Seção 4.3, apresentamos algumas estatísticas sobre os *traces* de pacotes que foram utilizados nos experimentos. O objetivo dessa análise estatística é destacar as principais diferenças dos *traces* utilizados. Na Seção 4.4, apesentamos uma avaliação isolada do Coletor, mostrando o impacto de algumas alterações que fizemos no algoritmo original, como também o impacto da amostragem no seu funcionamento. A avaliação da solução completa é apresentada na Seção 4.5. Finalmente, a Seção 4.6 traz as considerações finais desse capítulo.

4.1 Ambiente de Avaliação

Avaliamos o ODHIn em dois ambientes. No primeiro, utilizamos um simulador próprio desenvolvido a partir do código utilizado em Jose et al. [49], o qual nos foi gentilmente cedido pelos autores. Esse simulador processa *traces* reais de pacotes, sendo cada pacote tratado como um evento discreto. Além disso, implementamos no simulador as funcionalidades da aplicação OF e do Coletor.

No segundo ambiente, utilizamos a versão 2.0.0 da ferramenta de prototipação Mininet Hi-Fi [35]. Essa ferramenta é capaz de emular hosts virtuais, comutadores e os enlaces entre eles, ou seja, os mesmos componentes encontrados em um ambiente de rede real. A diferença, no entanto, é que esse emulador utiliza um comutador OpenFlow em software, o que de certa forma limita a escala dos experimentos realizados. No ambiente emulado, a aplicação OF foi implementada utilizando a versão *betta* do controlador POX [77]. O Coletor foi implementado como um dispositivo à parte do controlador, sendo conectado diretamente ao comutador OF. O Coletor foi implementado na linguagem C++ e utiliza um canal de controle para comunicação com a aplicação OF.

Em ambos os ambientes, utilizamos um fluxo de pacotes IP proveniente de um tráfego capturado de uma rede real. Dessa forma, simulamos o cenário de uma rede de borda conectada à Internet. No ambiente emulado, o tráfego de pacotes foi injetado no comutador OpenFlow usando a ferramenta *tcpreplay*[1]. É importante observar que o ambiente emulado, apesar de simples, permitiu avaliar as funcionalidades do ODHIn, bem como mensurar sua acurácia.

Realizamos todos os experimentos, em ambos os ambientes, em uma máquina com processador Intel Core i5-3570K 3.40 GHz e memória RAM de 8 GB. A Tabela 4.1 apresenta os valores dos principais parâmetros utilizados nos experimentos descritos neste capítulo.

Parâmetro	Valores
limiar de detecção (ϕ)	$0,2$; $0,1$; $0,05$; $0,02$; $0,01$
intervalo de amostragem	1 μs, 10 μs, 50 μs, 100 μs, 500 μs, 1 ms, 5 ms, 10 ms
erro tolerado no Coletor (ε)	$0,001$
intervalo de medição (M)	5 s
intervalo de confiança	95 %

Tabela 4.1: *Parâmetros utilizados nos experimentos.*

Além dos ambientes descritos acima, realizamos também testes utilizando a infraestrutura de experimentação do FIBRE (*Future Internet testbeds/experimentation between BRazil and Europe*) [29]. Esses testes foram executados em dois momentos distintos: no início da implantação da ilha e, mais recentemente, como usuário experimentador da ilha. No início da implantação, realizamos os experimentos utilizando um servidor com uma placa NetFPGA [61] e quatro portas 1 Gbps. Esse servidor atuou como comutador OpenFlow, utilizando comutador em software padrão distribuido com a placa NetFPGA. No entanto, como esse software implementava apenas a versão 1.0 do protocolo OF, tivemos dificuldades em realizar os experimentos nesse ambiente, pois a versão 1.0 do OpenFlow não suporta o recurso de múltiplas tabelas de regras. Como usuário experimentador, também encontramos problemas para usar o recurso de múltiplas tabelas devido à incompatibilidade da aplicação OF com o arcabouço de virtualização dos recursos OpenFlow da ilha.

[1] http://tcpreplay.synfin.net

4.2 Métricas de Avaliação

4.2.1 *Recall*, *Precision* e *F-measure*

É comum trabalhos da literatura sobre agregações hieráquicas de fluxos [15, 49, 99], a utilização das métricas *recall* e *precision*, as quais são empregadas na área de recuperação de informação. De maneira genérica, a métrica *recall* é a relação entre a quantidade de itens corretamente identificados por um algoritmo não exato e a quantidade total de itens reportados por um algoritmo exato. A métrica *recall* representa o complemento dos itens não detectados e, portanto, quanto maior o valor da métrica, menor será a quantidade de falsos negativos. De forma similar, a métrica *precision* é a relação entre a quantidade de itens corretamente identificados por um algoritmo não exato e a quantidade total de itens reportados por esse mesmo algoritmo. A métrica *precision* representa o complemento dos itens que foram detectados erroneamente e, portanto, quanto maior o valor da métrica, menor será a quantidade de falsos positivos.

O cálculo das métricas *recall* e *precision* requer a identificação dos itens que são verdadeiramente corretos. Para isso, é utilizado um algoritmo de referência (ou *offline*) que provê as respostas exatas, de acordo com o problema em estudo. Normalmente, um algoritmo de referência calcula seus resultados utilizando várias passadas sobre os dados, permitindo que suas respostas possuam um alto grau de acurácia. No contexto desse trabalho, as métricas *recall* e *precision* estão sendo utilizadas para avaliar a acurácia da solução proposta e, portanto, os itens corretos correspondem aos prefixos HHHs que foram corretamente identificados. Assim, podemos formular as métricas da seguinte maneira:

$$R = \frac{qtde\ HHHs\ corretos}{qtde\ total\ HHHs\ alg.\ offline} \tag{4-1}$$

$$P = \frac{qtde\ HHHs\ corretos}{qtde\ total\ HHHs\ reportados}. \tag{4-2}$$

Para classificar adequadamente os prefixos HHHs retornados pelo ODHIn utilizamos um algoritmo exato proposto por Cormode et al. [17]. Esse algoritmo trabalha de maneira *offline*, realizando várias passadas sobre os dados de entrada para identificar os prefixos HHHs de maneira exata. Esse algoritmo foi escolhido pois emprega uma granulosidade no nível de bits, a mesma que está sendo utilizada no ODHIn, e também por tratar as agregações hierárquicas de maneira bidimensional.

Outra métrica também utilizada no campo de recuperação de informação é a *F-measure* [11, 80], a qual é definida como:

$$F = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R}, \tag{4-3}$$

onde P é *precision*, R é *recall* e β é a importância relativa de *recall* sobre *precision*. No contexto deste trabalho, estamos interessados em *recall* e *precision* com pesos iguais, ou seja, $\beta = 1$. Dessa forma, a métrica *F-measure* pode ser expressa como:

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}. \tag{4-4}$$

A métrica *F-measure* assume valores no intervalo de $[0..1]$. Valores próximos de 1 significam resultados mais acurados, uma vez que essa métrica representa a composição de *recall* e *precision*.

4.2.2 *Offline similarity metric*

Nas avaliações iniciais do ODHIn percebemos que os resultados retornados pelo Coletor possuíam uma pequena diferença, em bits, no comprimento dos prefixos reportados em relação ao algoritmo exato. Esse fato ocorre pois o Coletor utiliza um algoritmo aproximado e, portanto, suas respostas são similares àquelas retornadas pelo algoritmo exato, mas não são iguais. Como as métricas *recall* e *precision* consideram apenas combinações exatas, elas resultam em valores numéricos baixos, dificultando a avaliação de acurácia da solução.

Essa dificuldade nos motivou a desenvolver uma métrica mais adequada para avaliar a qualidade das respostas de um mecanismo de detecção de HHHs. A nova métrica é chamada *Offline similarity metric* (O_{sm}) e representa uma medida de similaridade entre o conjunto de prefixos reportados pela solução e o conjunto retornado pelo algoritmo exato. A métrica O_{sm} é baseada em uma função de similaridade, a qual utiliza uma média ponderada para efetuar uma comparação bit a bit entre dois prefixos. Para calcular a função de similaridade, utilizamos um vetor de pesos, cujo intuito é priorizar os bits mais significativos do prefixo. Dado dois prefixos A e B de comprimento n e um vetor de pesos decrescente $W = [w_1, w_2, w_3, ..., w_n]$, a função de similaridade entre A e B é definida como:

$$S(A,B) = \frac{\sum_{i=1}^{n} s_i \cdot w_i}{\sum_{i=1}^{n} w_i}, \tag{4-5}$$

onde:

$$s_i = \begin{cases} 1, & se\ A_i = B_i \\ 0, & se\ A_i \neq B_i \end{cases} \tag{4-6}$$

Como os prefixos retornados pelos algoritmos são bidimensionais, o resultado final da função de similaridade é calculado através da média aritmética entre os valores de S correspondentes às partes origem e destino do prefixo, ou seja:

$$S = \frac{S_{origem} + S_{destino}}{2}. \tag{4-7}$$

Dado dois conjuntos de prefixos $X = \{x_1, x_2, x_3, ..., x_n\}$ e $Y = \{y_1, y_2, y_3, ..., y_n\}$, a métrica O_{sm} é definida como:

$$O_{sm}(X,Y) = \frac{\sum_{i=1}^{n} S_i(x_i, y_i)}{n}. \quad (4\text{-}8)$$

A métrica O_{sm} também assume valores no intervalo $[0..1]$ e seu valor representa a porcentagem de similaridade entre o conjunto de HHHs da solução avaliada e o conjunto reportado pelo algoritmo exato.

4.3 Análise Estatística dos *Traces* Reais

Nessa seção apresentamos as características mais relevantes acerca dos *traces* de pacotes utilizados nas avaliações do ODHIn. Utilizamos dois conjuntos de *traces*. O primeiro contém um tráfego de pacotes capturado a partir de uma rede real, enquanto o segundo é formado por um tráfego de pacotes gerado artificialmente. Para os *traces* com pacotes reais, utilizamos dois tipos de tráfego: 1) enlace de 1 Gbps no roteador de borda da UFG, no qual se conecta à Internet; 2) enlace OC192 disponibilizado pelo projeto CAIDA (*Center for Applied Internet Data Analysis*) [9]. Esses *traces*, denominados respectivamente Campus e CAIDA, permitem avaliar o comportamento do ODHIn em um cenário similar ao que é encontrado nas redes de núcleo do mundo real. Nas Seções 4.3.1 e 4.3.2 apresentamos algumas estatísticas que mostram características dos dois tipos de tráfego. Outras características são apresentadas na Tabela 4.2.

	Campus	CAIDA
Quantidade total de pacotes	8,5 M	169 M
Volume de tráfego	5,8 GB	141 GB
Tamanho médio dos pacotes	689 bytes	832 bytes
Vazão	155 Mbps	3,7 Gbps
Taxa de pacotes	28 Kpps	564 Kpps
Quantidade de fluxos	532 K	4,9 M
Taxa de fluxos	1,7 K	16,6 K
Duração	300 s	300 s

Tabela 4.2: *Algumas características dos traces reais utilizados.*

Utilizamos o segundo conjunto, com *traces* sintéticos, para avaliar a resposta do ODHIn em cenários específicos de tráfego. Para isso, geramos *traces* sintéticos com dois perfis distintos: 1) tráfego constante, no qual o conjunto de HHHs permanece o mesmo durante todo o experimento; 2) tráfego variável, no qual o conjunto de HHHs muda entre os intervalos de medição. Os pacotes gerados nos *traces* sintéticos são provenientes de prefixos que são classificados como HHHs. A lista de prefixos HHHs foi gerada de maneira aleatória e a quantidade de prefixos segue a relação $1/\phi$.

4.3.1 Distribuição de Pacotes

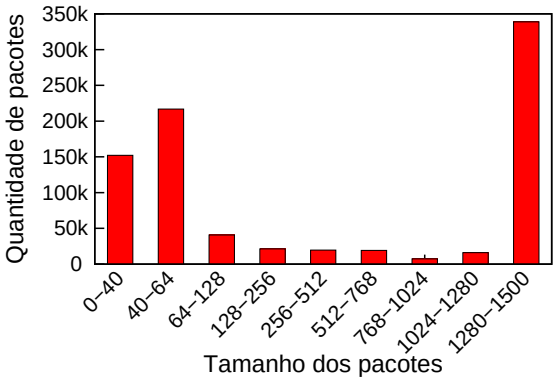
Para entender melhor as diferenças entre os *traces* reais utilizados, apresentamos inicialmente a distribuição por tamanho dos pacotes. Analisamos a distribuição levando em conta todos os pacotes IP presentes nos *traces* e empregando os seguintes intervalos: $0-40$, $40-64$, $64-128$, $128-256$, $256-512$, $512-768$, $768-1024$, $1024-1280$, $1280-1500$. Além disso, extraímos a distribuição em três partes distintas dos *traces*, a saber: início, meio e fim. Em cada uma das partes foi considerada uma janela com duração de 30 segundos de tráfego. Adicionalmente, extraímos também a distribuição correspondente ao *trace* completo. A Figura 4.1 mostra a distribuição para o *trace* Campus e a Figura 4.2 apresenta a distribuição para o *trace* CAIDA.

Para os dois *traces* percebemos uma predominância de pacotes grandes, com até 1500 bytes. Normalmente, esse tipo de pacote é empregado por aplicações que transmitem suas informações utilizando um protocolo de transporte confiável. Nessa faixa do tráfego estão localizados os *hosts* que consomem a maior parte dos recursos da rede, os quais representam as agregações hierárquicas de fluxos. No *trace* CAIDA observamos uma porcentagem significativa de pacotes nessa faixa, atingindo quase 50 % do total de tráfego. As faixas $0-40$ e $40-64$, que ocupam o segundo e terceiro lugar respectivamente, representam os pacotes de confirmação de recebimento usado em protocolos de transporte.

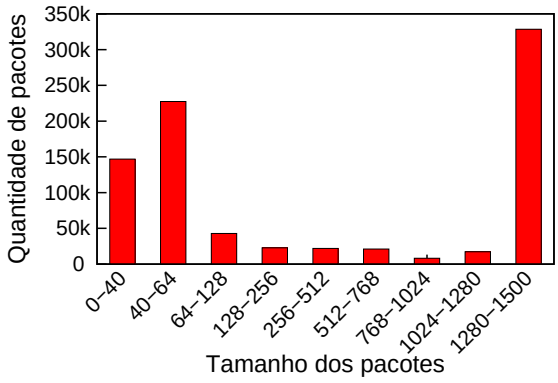
4.3.2 Distribuição das Agregações Hierárquicas

No contexto de monitoramento é importante entender quais características presentes em um tráfego de pacotes que afetam a acurácia de uma solução para detecção de agregações hierárquicas de fluxos. Para isso, apresentamos a distribuição com os comprimentos dos prefixos das agregações hieráquicas bidimensionais referente aos *traces* reais utilizados. A distribuição foi obtida através da análise dos prefixos HHHs, detectados pelo ODHIn, e consideramos dois intervalos para cada octeto do endereço IP. Além disso, empregamos os seguintes parâmetros na análise: intervalo de medição $M = 5s$, total de tráfego de 5 minutos, limiar de detecção $\phi = 0,01$ e intervalo de amostragem de $1\mu s$. A distribuição para o *trace* Campus é mostrada na Figura 4.3a e a distribuição para o *trace* CAIDA é mostrada na Figura 4.3b.

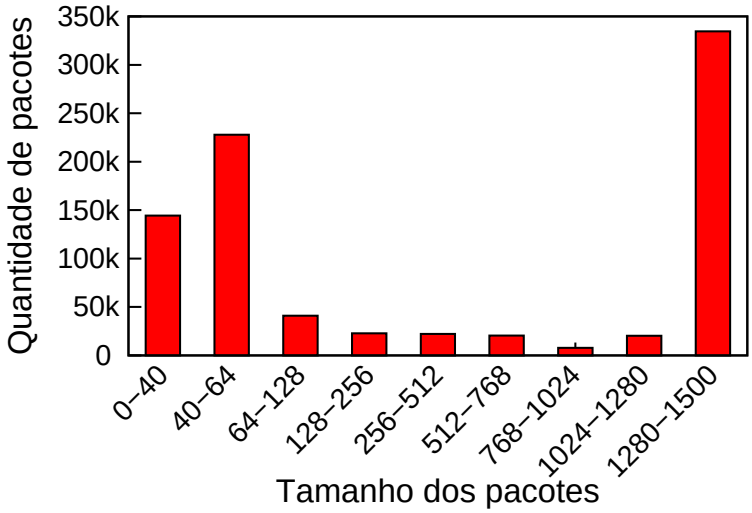
Observando a Figura 4.3, podemos notar claramente algumas diferenças entre os dois tipos de tráfego. Primeiramente, há uma predominância de agregações de 4 bits no *trace* CAIDA, enquanto no *trace* Campus percebemos uma distribuição de prefixos mais variada. A razão disso é que o *trace* CAIDA representa um tráfego proveniente de vários sistemas autônomos diferentes e, portanto, possui um alto nível de dispersão dos endereços IP. Por esse motivo, a maioria das agregações hierárquicas é formada nos bits



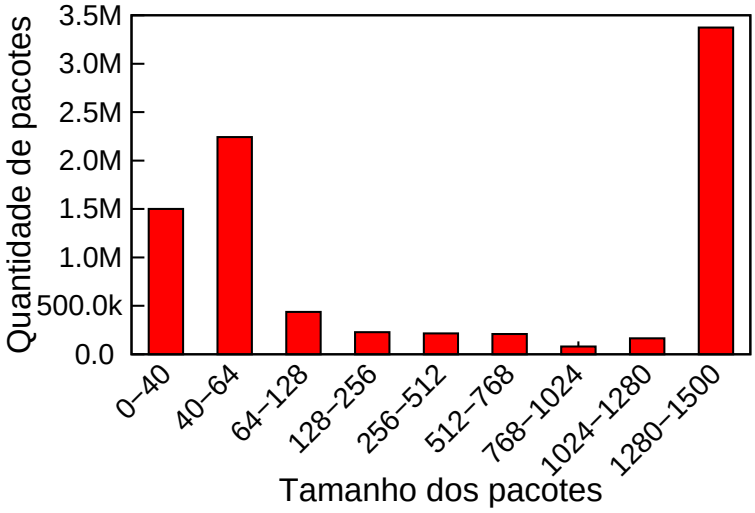
(a) Início do *trace*.



(b) Metade do *trace*.

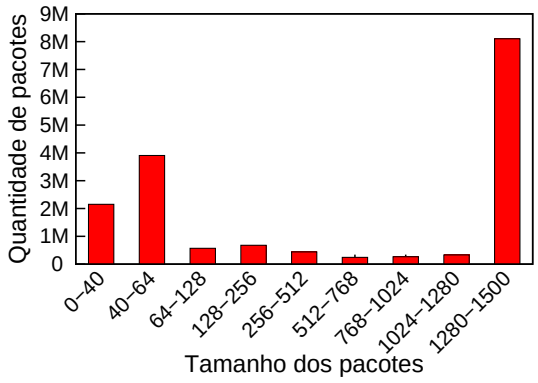


(c) Parte final do *trace*.

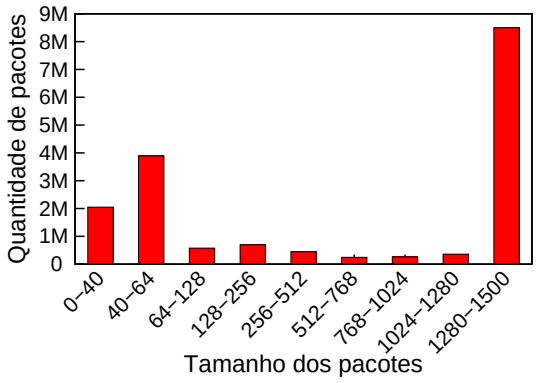


(d) *Trace* completo.

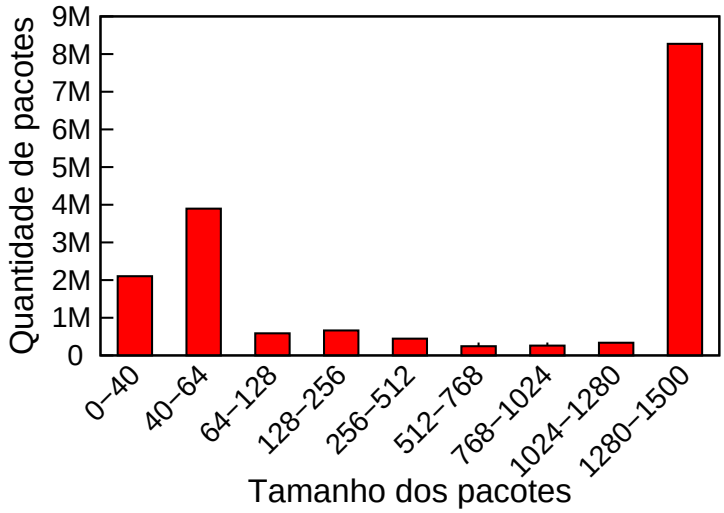
Figura 4.1: *Histograma com a distribuição do tamanho dos pacotes para o trace Campus.*



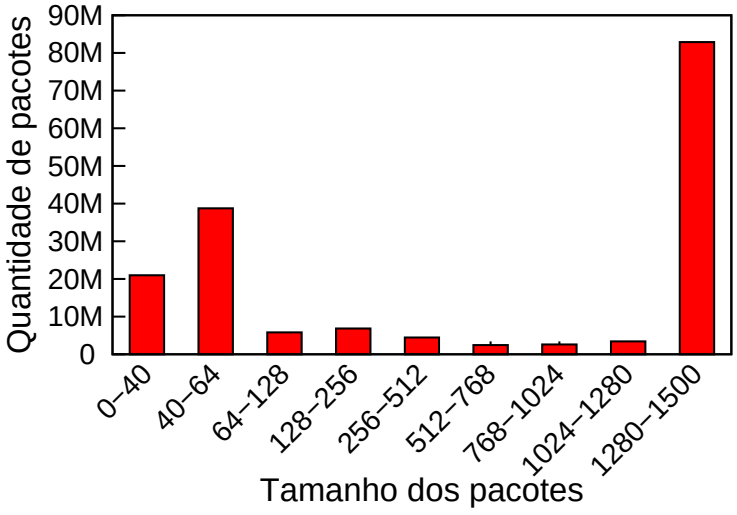
(a) Início do *trace*.



(b) Metade do *trace*.

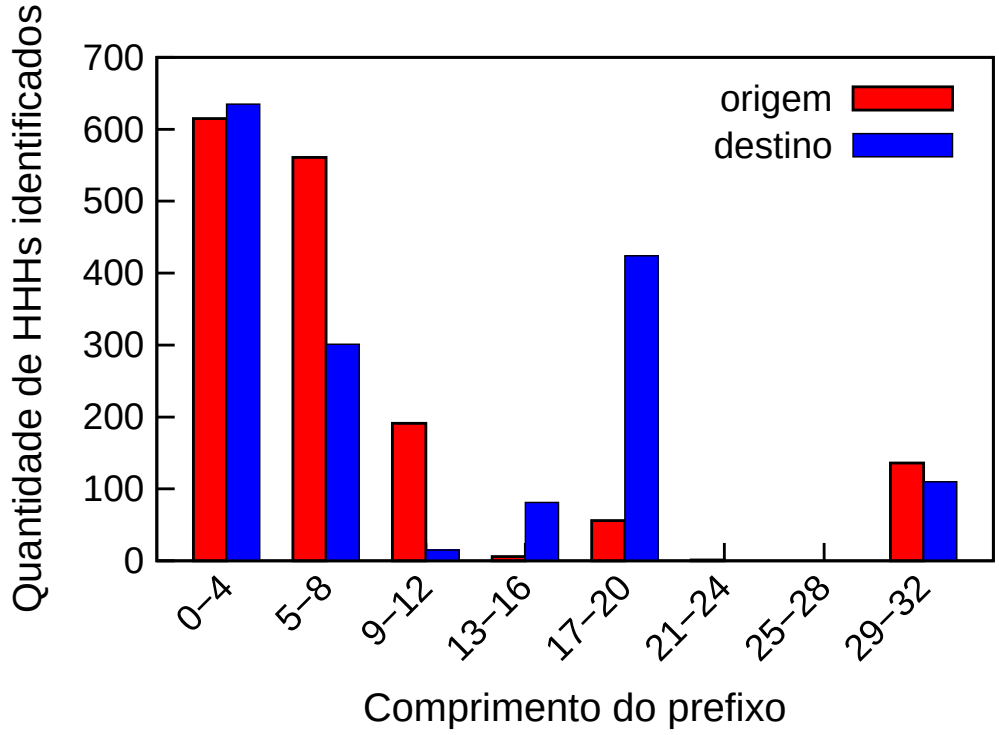


(c) Parte final do *trace*.

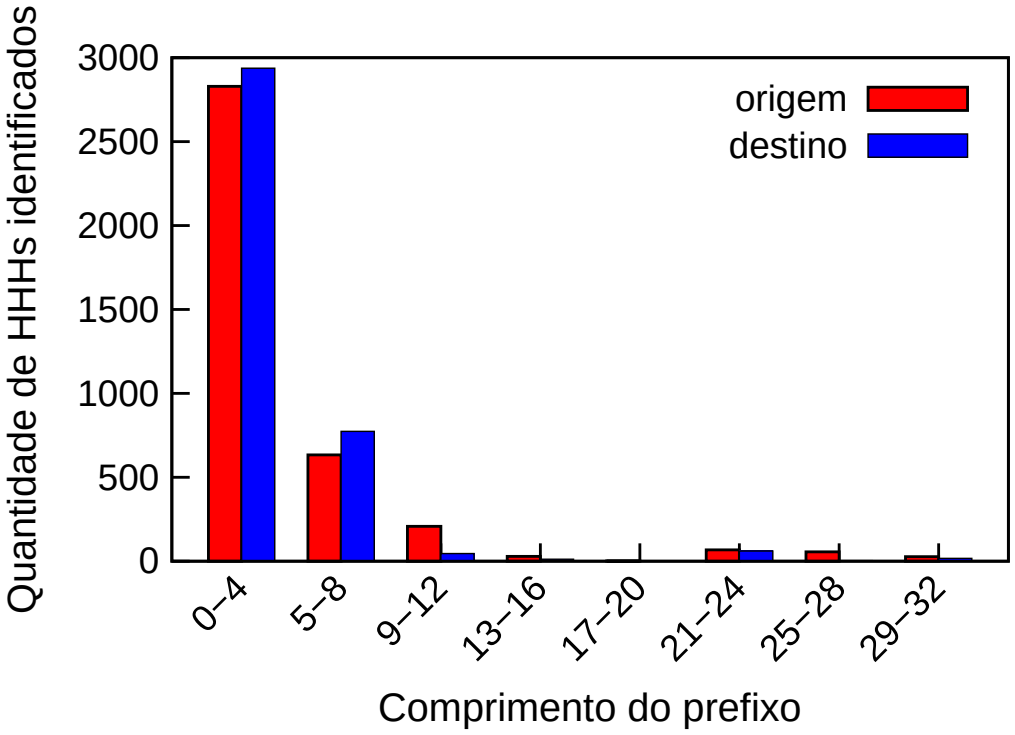


(d) *Trace* completo.

Figura 4.2: *Histograma com a distribuição do tamanho dos pacotes para o trace CAIDA.*



(a) *Trace* Campus.



(b) *Trace* CAIDA.

Figura 4.3: *Histograma com a distribuição do comprimento dos prefixos HHHs identificados.*

iniciais do espaço de endereçamento. É importante ainda ressaltar, um comportamento particular que ocorre no *trace* Campus no intervalo de $17-20$, no qual identificamos mais de 420 prefixos de destino. Esse intervalo coincide com o bloco de endereçamento

IPv4 da instituição, o qual é um $/19$ e, portanto, muitas das agregações identificadas são formadas nesse intervalo.

4.4 Avaliação do Coletor

Nesta seção apresentamos uma avaliação do componente Coletor. Na Seção 4.4.1 avaliamos o efeito de uma modificação realizada no algoritmo original do Coletor. Na Seção 4.4.2 apresentamos uma avaliação do impacto do mecanismo de amostragem sobre seu funcionamento.

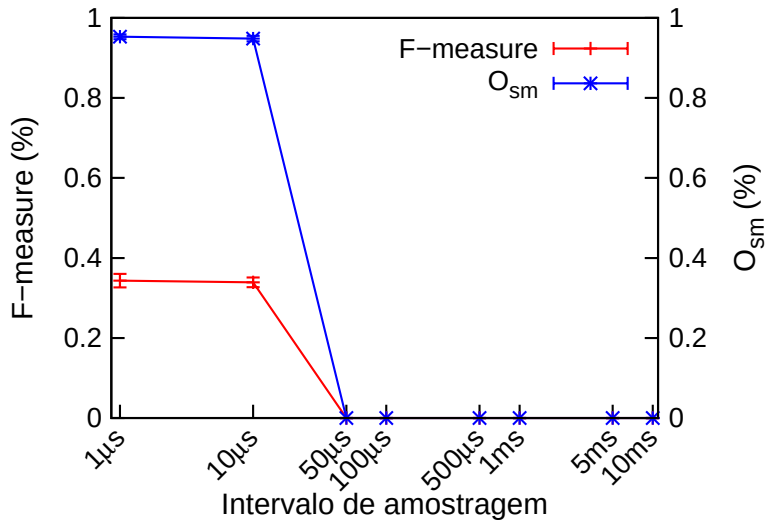
4.4.1 Preenchimento das tabelas *hash*

Nesta seção apresentamos resultados que mostram os efeitos práticos de uma modificação no algoritmo original do Coletor. Essa modificação teve como objetivo corrigir um comportamento inesperado observado quando o algoritmo recebe uma baixa quantidade de pacotes. Esse comportamento interfere tanto no funcionamento quanto na acurácia do Coletor.

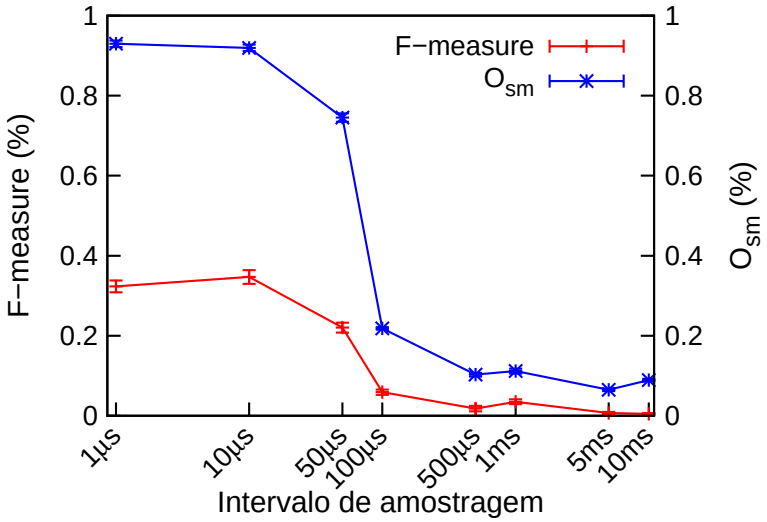
Para avaliar os efeitos da modificação implementada, preparamos dois conjuntos de experimentos. No primeiro, utilizamos *traces* sintéticos sendo injetados no Coletor a uma taxa constante. Os *traces* foram gerados de maneira aleatória e, portanto, a relação de prefixos HHHs é conhecida a priori. O limiar de detecção $\phi = 0,02$ foi utilizado na geração dos *traces*. Empregamos diferentes valores para o intervalo de amostragem, simulando uma diminuição gradual na quantidade de pacotes recebidos pelo algoritmo. Em cada experimento, utilizamos 30 *traces* e os valores médios foram calculados com um intervalo de confiança de 95 %.

Os gráficos com as métricas de acurácia são mostrados na Figura 4.4. Podemos observar que ocorre um decréscimo acentuado na acurácia, em ambas as métricas, com o algoritmo original. Esse fato ocorre porque o algoritmo não consegue reportar adequadamente os prefixos HHHs correspondentes ao tráfego amostrado, em especial quando são empregados intervalos de amostragem acima de 10 μs. A razão disso é que no algoritmo original as tabelas *hash* intermediárias da matriz H não são populadas devido à expansão muito rápida das *tries* unidimensionais. No algoritmo modificado podemos notar que esse comportamento não ocorre pois as tabelas da matriz H são populadas corretamente, mesmo quando são utilizadas baixas taxas de amostragem. Nesse caso, a diminuição de acurácia é uma consequência esperada pois o algoritmo está fornecendo respostas com base em uma quantidade menor de amostras.

O segundo conjunto de experimentos tem a finalidade de avaliar a ocupação das tabelas *hash* nos dois algoritmos, além da influência no valor do parâmetro k na versão



(a) Algoritmo original.



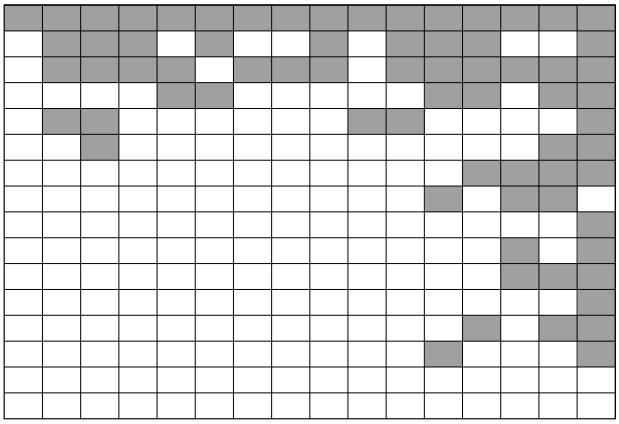
(b) Algoritmo modificado com $k = 1$.

Figura 4.4: *Efeito das modificações feitas no algoritmo do Coletor, utilizando traces sintéticos.*

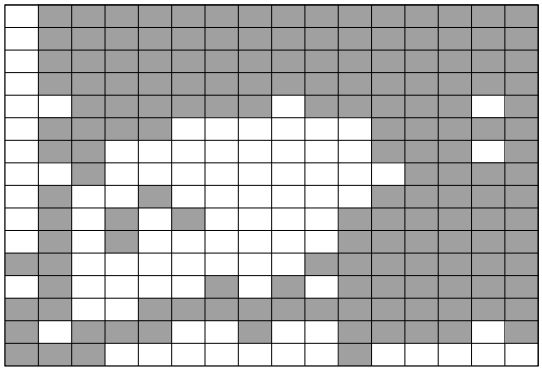
modificada. Para isso, utilizamos o *trace* Campus, que representa um tráfego capturado de uma rede real. Utilizamos também o valor de 500 μs como intervalo de amostragem. É importante ressaltar que o resultado desse experimento independe do valor do limiar de detecção, pois o algoritmo não utiliza o valor de ϕ como critério para alimentar a matriz H. O parâmetro k do algoritmo modificado foi gerado a partir de uma série exponencial 2^n, com valores de $n = \{0,1,2,3\}$.

Na Figura 4.5 é mostrada uma representação visual da matriz H para ambos os algoritmos. Nessa representação, os elementos em branco significam tabelas *hash* vazias e os elementos em cinza significam tabelas *hash* populadas. Podemos observar que no algoritmo original uma parte significativa das tabelas não são populadas. Isso acontece devido à rápida expansão das estruturas unidimensionais do algoritmo, as quais

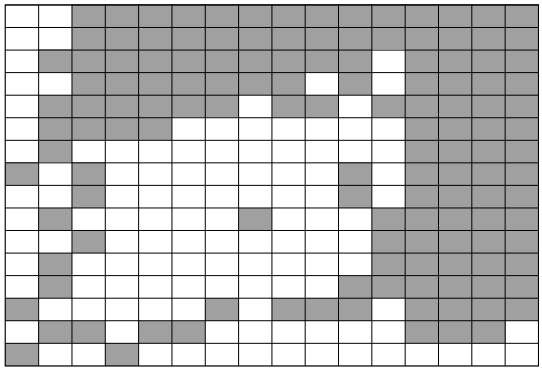
são utilizadas como base para alimentar as tabelas da matriz H. Como consequência, na etapa subsequente, de reconstrução dos contadores da matriz e cálculo dos HHHs, o algoritmo não consegue identificar todos os prefixos correspondentes às agregações dos pacotes amostrados. Por outro lado, no algoritmo modificado, o preenchimento das tabelas *hash* acontece de maneira mais esparsa, sobretudo com $k = 1$, como pode ser visto na Figura 4.5b.



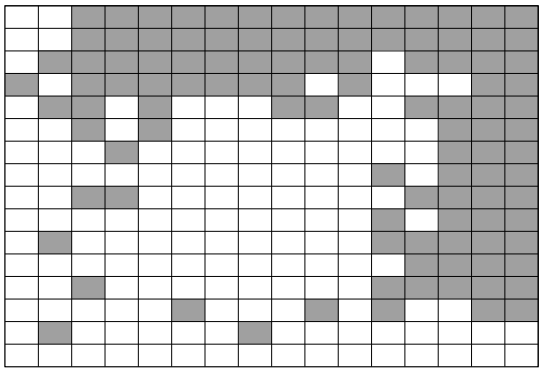
(a) Algoritmo original.



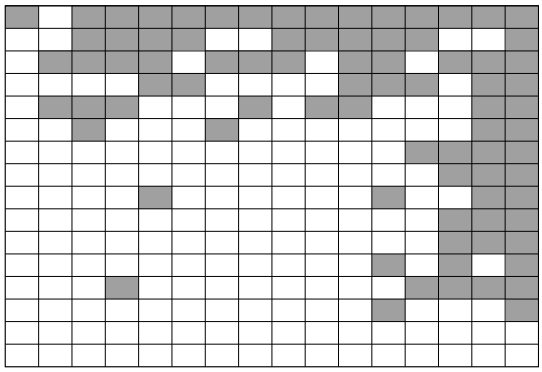
(b) Algoritmo modificado com $k = 1$.



(c) Algoritmo modificado com $k = 2$.



(d) Algoritmo modificado com $k = 4$.



(e) Algoritmo modificado com $k = 8$.

Figura 4.5: *Ocupação das tabelas hash intermediárias da matriz H, no algoritmo do Coletor.*

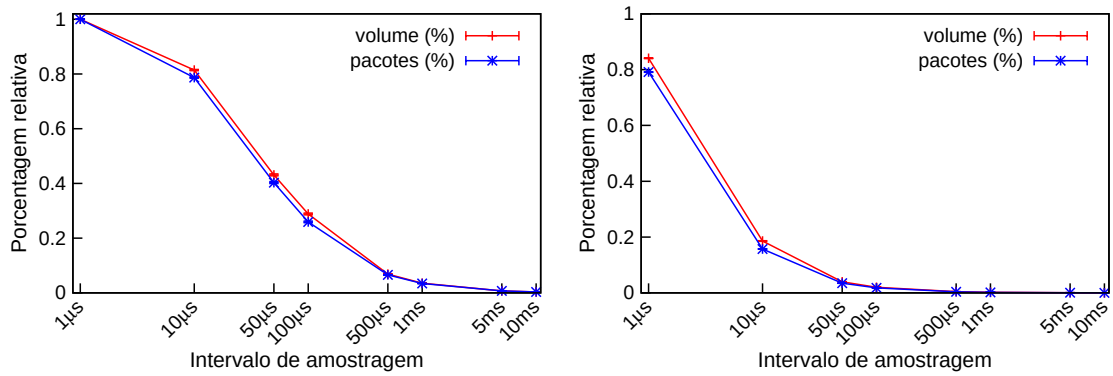
Nesse algoritmo, as agregações hierárquicas são identificadas de forma mais adequada devido à modificação no retorno das estruturas unidimensionais, a qual permite que um maior número de tabelas sejam preenchidas. À medida que aumentamos o valor de k, observamos que o algoritmo modificado tende a exibir um comportamento semelhante ao algoritmo original. De fato, se utilizarmos $k = 32$ devemos obter um comportamento muito parecido nos dois algoritmos.

4.4.2 Impacto da Amostragem

Nesta seção, apresentamos uma avaliação do impacto da amostragem no Coletor. A avaliação é dividida em duas partes, uma quantitativa e outra qualitativa. Na avaliação quantitativa, mostramos como o nível de informação reduz, à medida que o intervalo de amostragem aumenta. Na avaliação qualitativa, apresentamos resultados de acurácia medidos a partir das respostas providas diretamente pelo Coletor, de maneira isolada do restante da solução.

A Figura 4.6 mostra, para os *traces* reais, a quantidade percentual de volume e de pacotes correspondentes às amostras coletadas em relação ao total de tráfego que foi enviado para o Coletor. No *trace* Campus, observamos que o intervalo de amostragem de 1 μs consegue preservar as características originais do tráfego, pois constatamos que quase 100 % do tráfego total é capturado nas amostras. Para valores maiores do intervalo de amostragem, percebemos uma queda significativa no percentual amostrado, tanto de volume quanto de pacotes. As reduções mais substanciais ocorrem com intervalos de 10 μs e 50 μs com quedas percentuais de aproximadamente 20 % e 40 %, respectivamente. Para o *trace* CAIDA, observamos um cenário diferente. No intervalo de 1 μs, percebemos uma queda de quase 20 % em relação ao tráfego original e, no intervalo seguinte, a queda é ainda mais drástica, aproximadamente 60 % se comparado com o intervalo anterior. Esse comportamento é explicado pelo fato do *trace* CAIDA ter sido capturado em um enlace que interconecta múltiplas redes de núcleo, representando condições muito intensas de tráfego, na qual observamos intervalos de chegada de pacotes inferiores a 1 μs. Nessas condições, é esperado que a utilização da amostragem reduza a qualidade das respostas providas pelo Coletor, pois a técnica empregada diminui substancialmente a quantidade de informação disponível para o algoritmo.

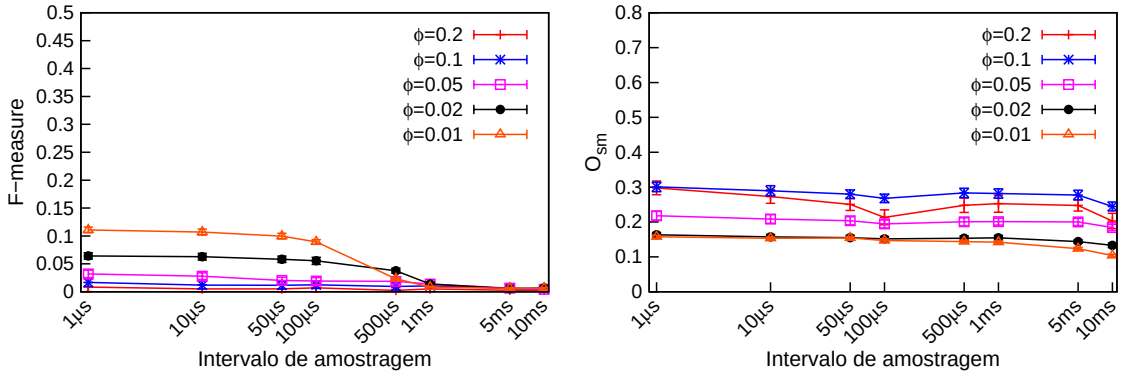
Na segunda avaliação, apresentamos os resultados referentes às métricas de acurácia considerando diferentes valores de amostragem e variando o limiar de detecção ϕ. A Figura 4.7 mostra os resultados das métricas *F-measure* e O_{sm} para o *trace* Campus, enquanto a Figura 4.8 mostra os resultados das mesmas métricas para o *trace* CAIDA. No *trace* Campus, podemos notar que os valores de *F-measure* se mantêm quase constantes quando utilizamos intervalos de até 50 μs. Esse resultado é explicado pelo fato das



(a) *Trace* Campus. **(b)** *Trace* CAIDA.

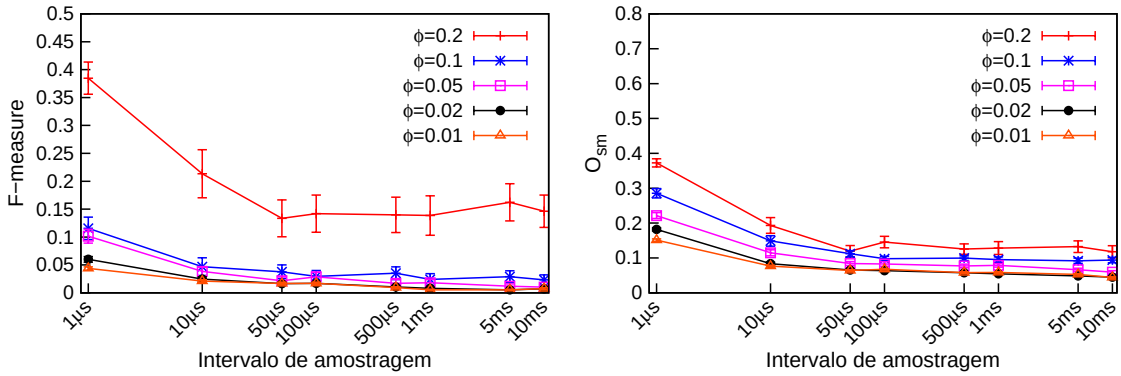
Figura 4.6: *Impacto da amostragem sobre o volume e a quantidade de pacotes recebidos no Coletor.*

agregações identificadas nesse tráfego serem formadas por fluxos pertencentes a uma faixa limitada de endereços IP. No *trace* CAIDA, os melhores resultados de acurácia ocorrem com limiares de detecção maiores, como por exemplo, $\phi = 0,2$. Nesse caso, como o tráfego está disperso em uma grande faixa de endereços, o algoritmo do Coletor tem maior dificuldade de reportar corretamente os HHHs, sendo mais acurado nos cenários onde a quantidade e o comprimento dos prefixos reportados é menor. Isso ocorre quando os valores de ϕ são maiores. Em ambos os *traces*, como era esperado, observamos uma tendência de diminuição da acurácia nas respostas do Coletor à medida que o intervalo de amostragem aumenta.



(a) Métrica *F-measure*. **(b)** Métrica O_{sm}.

Figura 4.7: *Impacto da amostragem sobre a acurácia do Coletor com o trace Campus.*



(a) Métrica *F-measure*. **(b)** Métrica O_{sm}.

Figura 4.8: *Impacto da amostragem sobre a acurácia do Coletor com o trace CAIDA.*

4.5 Avaliação do ODHIn

Nesta seção, apresentamos os resultados dos experimentos que realizamos com o ODHIn, tendo como propósito avaliar a solução completa nos quesitos de funcionalidade e acurácia. Esta seção está organizada da seguinte forma. A Seção 4.5.1 descreve os experimentos realizados no ambiente simulado, enquanto a Seção 4.5.2 apresenta os resultados da avaliação no ambiente emulado.

4.5.1 Avaliação em Ambiente Simulado

Apresentamos os resultados do ODHIn em comparação com o algoritmo proposto por Jose et al. [49]. Esse algoritmo identifica as agregações hierárquicas unidimensionais presentes em um tráfego de pacotes, tendo também sido projetado para operar em redes OpenFlow. Os autores desse algoritmo disponibilizaram o código-fonte original da implementação, o qual foi usado nas avaliações comparativas. O algoritmo proposto por Jose et al. funciona da seguinte maneira. Duas listas de prefixos, provenientes de uma árvore, são mantidas com o propósito de monitoramento. A primeira lista é formada a partir da raiz da estrutura, enquanto a segunda lista representa os nós filhos dos prefixos da primeira lista. A cada intervalo de medição, o algoritmo verifica quais nós alcançaram o limiar de detecção, efetuando as ramificações dos elementos da estrutura e atualizando as listas de prefixos monitorados. Esse algoritmo emprega o contador de pacotes no cálculo das agregações hierárquicas. Utilizaremos a designação *hhh*1*D* para esse algoritmo, uma vez que são detectados apenas os prefixos unidimensionais. O pseudo-código do algoritmo *hhh*1*D* está apresentado no Apêndice A.

A avaliação comparando o ODHIn com o *hhh*1*D* foi realizada no ambiente simulado, utilizando os *traces* Campus e CAIDA. A Figura 4.9 mostra os resultados de *recall* e *precision* para o *hhh*1*D*, enquanto a Figura 4.10 mostra o resultado das

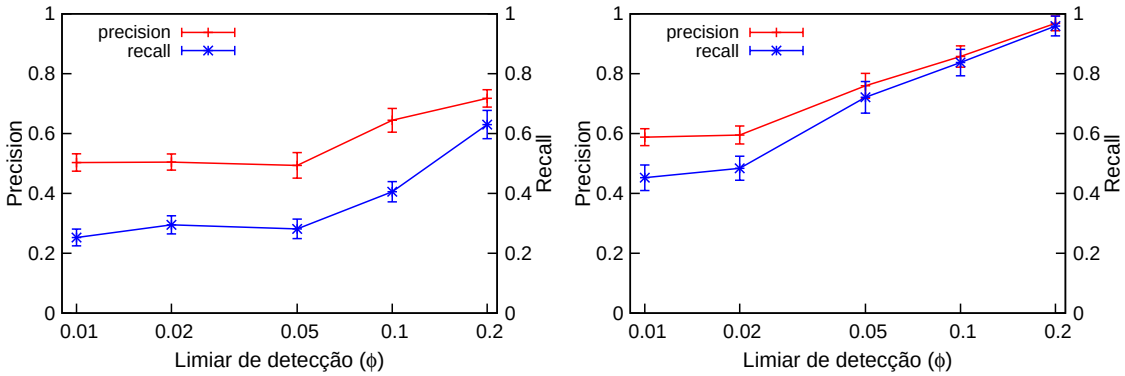
mesmas métricas para o ODHIn. Para o cálculo das métricas de acurácia do algoritmo *hhh*1*D*, utilizamos o algoritmo *offline* fornecido junto com o restante da implementação. O algoritmo *offline* empregado pelos autores consiste na mesma estratégia do algoritmo *online*, ou seja, o mesmo algoritmo base é utilizado para a identificação das agregações. A diferença é que o *offline* realiza várias passadas sobre o mesmo fluxo de pacotes até que o algoritmo atinja a convergência, ou seja, até que o conjunto de HHHs identificados permaneça inalterado.

Observando os resultados para o algoritmo *hhh*1*D*, podemos notar que, para ambos os *traces*, a acurácia aumenta com o aumento do limiar de detecção. Isso ocorre porque valores maiores de ϕ implicam em uma quantidade menor de prefixos identificados, resultando em menores taxas de falsos positivos. Outro detalhe, é que os *traces* utilizados nessa avaliação apresentam um perfil de tráfego constante, ou seja, o conjunto de HHHs não varia muito ao longo do tempo.

Em relação aos resultados do ODHIn, com o *trace* CAIDA, notamos um valor alto de *precision* com $\phi = 0,2$. A razão disso é que essa métrica é calculada em relação à quantidade de prefixos reportados pela nossa solução. No caso do ODHIn, essa quantidade sempre obedece a relação $1/\phi$. No entanto, o algoritmo exato utilizado nas avaliações do ODHIn reporta, na maioria das vezes, um quantitativo de prefixos acima de $1/\phi$ resultando em valores baixos para a métrica *recall*.

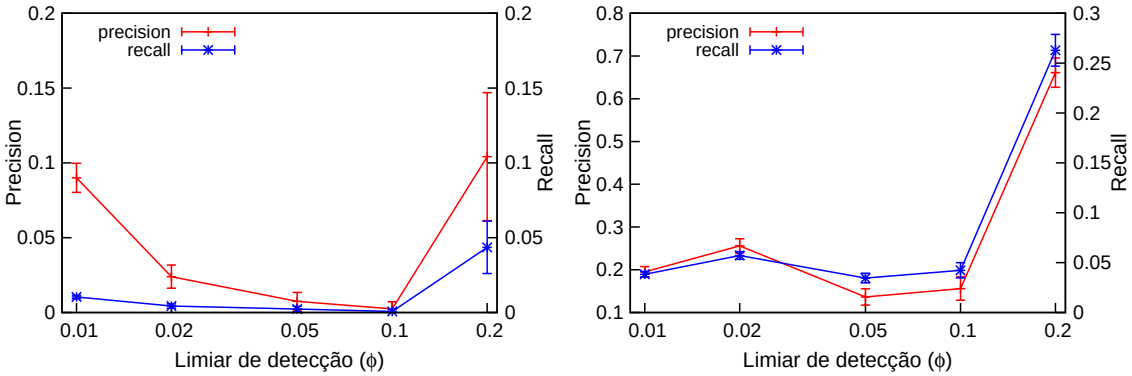
O segundo conjunto de avaliações apresenta os resultados entre os dois algoritmos com um *trace* sintético. Esse *trace* foi preparado para simular uma situação de tráfego variável, na qual o conjunto de agregações hierárquicas foi trocado a cada 30 segundos. O *trace* sintético foi gerado utilizando valor $\phi = 0,02$. Essa avaliação permitiu observar a resposta dos algoritmos diante de alterações frequentes no perfil de tráfego. A Figura 4.11 mostra os resultados de *recall* e *precison* para os *traces* Campus e CAIDA para o algoritmo *hhh*1*D*. A Figura 4.12 mostra os resultados para as mesmas métricas para o ODHIn. Nessa avaliação, percebemos uma melhor adaptação do ODHIn em relação às mudanças de tráfego. A nossa solução leva até 2 intervalos de medição para retornar ao patamar anterior de acurácia, enquanto o algoritmo *hhh*1*D* demora vários intervalos para se adaptar. A razão disso é que esse último algoritmo utiliza uma estrutura hierárquica pouco flexível, a qual leva alguns intervalos para se expandir de acordo com o perfil corrente de tráfego. No ODHIn, devido à utilização do Coletor, a estrutura converge de maneira mais rápida às mudanças de tráfego.

Uma outra vantagem do ODHIn é que a solução realiza o monitoramento de agregações de tráfego de maneira bidimensional. Dessa forma, nossa solução identifica os agregados hierárquicos de fluxos que atingem um dado limiar de detecção nas duas dimensões, endereço IP de origem e destino, simultâneamente. Isso é uma característica importante, uma vez que o monitoramento bidimensional consegue capturar adequada-



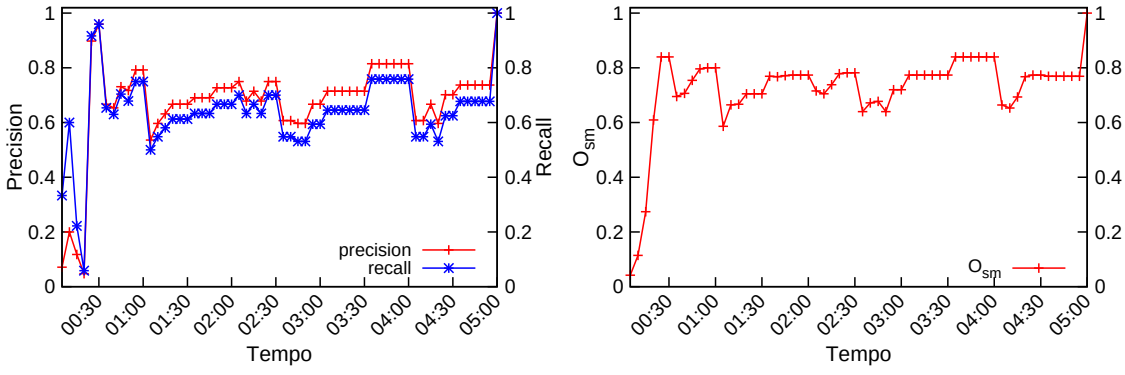
(a) *Trace* Campus. **(b)** *Trace* CAIDA.

Figura 4.9: *Gráfico de acurácia do algoritmo hhh*1*D*



(a) *Trace* Campus. **(b)** *Trace* CAIDA.

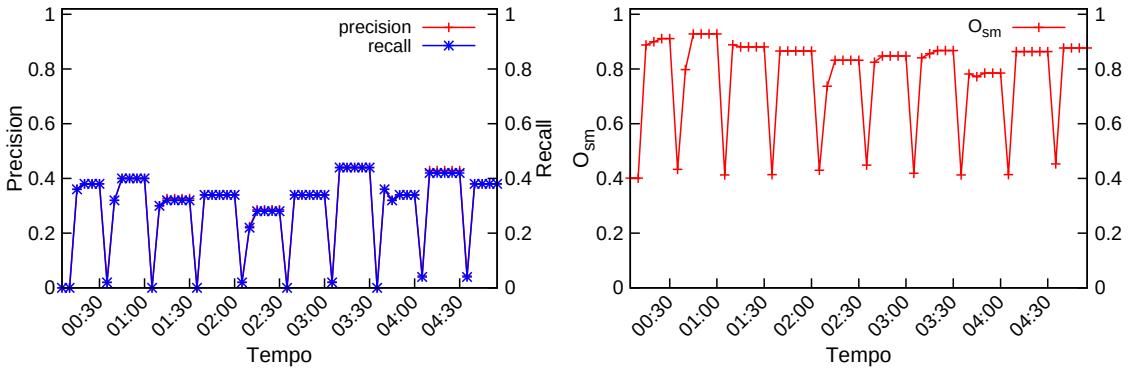
Figura 4.10: *Gráfico de acurácia do ODHIn.*



(a) Métricas *Recall* e *Precision*. **(b)** Nova métrica.

Figura 4.11: *Gráfico de acurácia do algoritmo hhh*1*D com trace sintético.*

mente alguns comportamentos de tráfego presentes em redes reais. Por exemplo, para monitorar a ocorrência de um ataque, o administrador necessita saber quais os pares de endereços mais frequentes no tráfego de pacotes, uma vez que esse tipo de atividade pode



(a) Métricas *Recall* e *Precision*.

(b) Nova métrica.

Figura 4.12: *Gráfico de acurácia do ODHIn com trace sintético.*

ocorrer em qualquer uma das dimensões monitoradas. As soluções anteriores do tipo unidimensional [49] falham em capturar esse tipo de comportamento, pois essas soluções possibilitam monitorar apenas uma dimensão por vez.

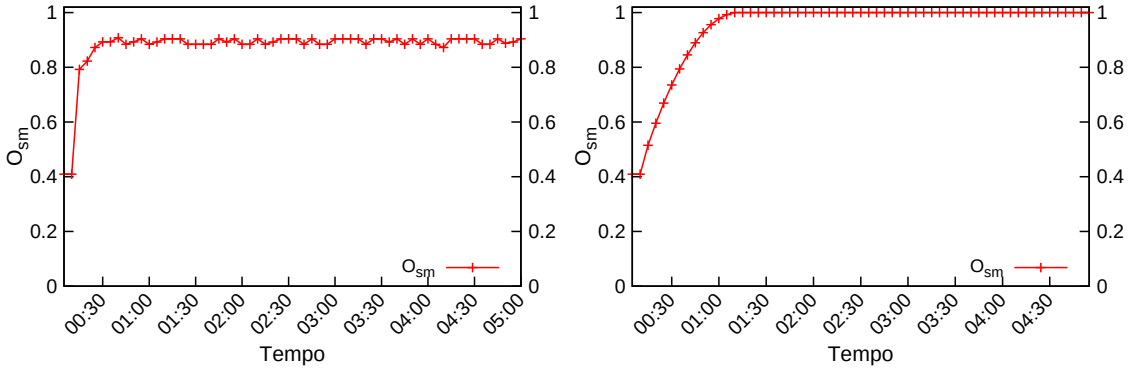
4.5.2 Avaliação em Ambiente Emulado

Nesta seção apresentamos uma avaliação do ODHIn sendo executado no ambiente emulado. Esse ambiente consiste de um comutador OpenFlow, do controlador e dois *host* adicionais conectados ao comutador. O primeiro *host* foi utilizado para executar uma instância do Coletor. O segundo *host* tem a única função de injetar um tráfego de pacotes no comutador.

Nesse experimento utilizamos um *trace* sintético com um perfil de tráfego constante, ou seja, o conjunto de agregações hierárquicas não varia ao longo do tempo. Para efetuar a comparação de convergência utilizamos a solução completa do ODHIn, e uma versão da solução com o componente Coletor desabilitado. Dessa forma, podemos avaliar os ganhos obtidos pela utilização desse componente. Na versão com o Coletor desativado, a expansão da árvore é realizada ao final de cada intervalo de medição e apenas nos prefixos que alcançaram o limiar de detecção. No experimento utilizamos a seguinte parametrização: limiar de detecção $\phi = 0,01$ e intervalo de amostragem de 1 μs. Nessa avaliação, estamos interessados no tempo de resposta da solução completa e, por isso, assumimos que o processamento adicional no Coletor, causado pela alta taxa de amostragem, pode ser negligenciado.

A Figura 4.13 mostra os resultados da avaliação com a métrica O_{sm}. Podemos perceber que a solução completa apresenta um menor de tempo de resposta, com cerca de 2 intervalos de medição para efetuar a adaptação de sua estrutura. Enquanto a solução com o Coletor desativado, apresenta um tempo maior para expandir sua estrutura, reportando uma acurácia adequada somente após 1 minuto de tráfego. Nessa versão, essa quantidade

de intervalos é suficiente para que ocorra a expansão da árvore e os nós consigam alcançar o comprimento dos prefixos que foram gerados no *trace* sintético, resultando em 100 % de acurácia. Na solução completa, as respostas fornecidas pelo Coletor possuem um erro associado, por se tratar de um algoritmo aproximado e, portanto, a solução completa não consegue atingir o mesmo patamar de acurácia que a versão sem esse componente.



(a) Coletor ativado. **(b)** Coletor desativado.

Figura 4.13: *Efeito do Coletor no ODHIn com trace sintético.*

4.6 Conclusão

Neste capítulo apresentamos os resultados da avaliação experimental do ODHIn. Apresentamos uma análise dos *traces* reais utilizados na avaliação, mostrando as diferenças entre os dois tipos de tráfego. Apresentamos resultados que comprovam os benefícios introduzidos pelas modificações no algoritmo original do Coletor e mostramos uma avaliação que demonstra o impacto da amostragem nas respostas desse componente. Ainda na parte de resultados, descrevemos os experimentos realizados com o ODHIn em comparação com os resultados obtidos com uma solução anterior da literatura. E por fim foram apresentados, os resultados referentes a solução proposta em um cenário que emula o ambiente real de uma rede SDN.

No próximo capítulo, discutiremos as principais contribuições deste trabalho, apontando as vantagens da sua utilização no monitoramento de agregações hierárquicas. Comentaremos ainda alguns pontos que foram identificados como perspectivas de trabalhos futuros.

CAPÍTULO 5

Conclusão e Trabalhos Futuros

Neste trabalho, abordamos uma solução de monitoramento desenvolvida para ser empregada em Redes Definidas por Software (SDN). As redes SDN representam um novo paradigma que propõe flexibilizar a forma de gerenciar, operar e monitorar as redes atuais através do desacoplamento do plano de dados e do plano de controle. O foco principal deste trabalho foi a pesquisa e o desenvolvimento de uma solução para detecção *online* de agregações hierárquicas bidimensionais de fluxos, que possa operar em conjunto com as redes definidas por software. Esse tema foi escolhido pois durante o levantamento bibliográfico se vislumbrou a possibilidade de realizar contribuições para a área, uma vez que foram identificados poucos trabalhos que aliam monitoramento de agregações hierárquicas aplicado às redes definidas por software.

No início do desenvolvimento deste trabalho, percebeu-se que a adoção de uma solução híbrida para o problema de identificação das agregações seria um caminho promissor. Dessa forma, foi projetada uma solução que pudesse aproveitar as vantagens de duas técnicas distintas: monitoramento flexível através de contadores e inspeção profunda de pacotes. Assim, a solução apresentada nesta dissertação possui componentes que realizam esses papéis. A solução é chamada de ODHIn, e se propõe a detectar rapidamente as agregações hierárquicas presentes em um fluxo de pacotes que esteja trafegando em um enlace de alta capacidade. Para isso, a solução é composta por dois componentes, sendo o primeiro responsável por realizar a contabilização do tráfego em uma estrutura hierárquica e efetuar a interação com os comutadores OpenFlow através da instalação e remoção de regras de monitoramento. O segundo componente tem a função de realizar uma inspeção profunda em uma pequena fração do tráfego de pacotes, possibilitando que as agregações sejam identificadas de forma mais ágil.

O presente trabalho envolveu o desenvolvimento das seguintes atividades:

- Estudo das técnicas e algoritmos existentes na área de detecção de agregações hierárquicas voltadas para redes convencionais.
- Levantamento e estudo de trabalhos envolvendo monitoramento em SDN de forma generalizada e monitoramento de agregações hierárquicas em específico.

- Proposta e desenvolvimento de uma solução híbrida que pudesse tirar proveito da flexibilidade provida pelas redes SDNs, combinada com técnicas convencionais que pudessem suprir algumas de suas restrições.
- Validação da solução proposta usando *traces* de pacotes com diferentes perfis de tráfego, utilizando para isso metodologias de simulação e de emulação.
- Comparação da solução proposta com outro trabalho existente na literatura.

Durante o desenvolvimento deste trabalho foram enfrentadas as seguintes dificuldades:

- O aprendizado com a ferramenta de desenvolvimento para aplicações OpenFlow utilizada nesse trabalho, o POX, exigiu muito tempo de estudo devido à pouca documentação disponível sobre sua interface de programação.
- A avaliação de acurácia da nossa solução requer a utilização de um algoritmo exato. O processo de escolha desse algoritmo foi demorado, pois tivemos dificuldade de encontrar soluções bidimensionais que operassem na granulosidade de bits.
- A replicação das mesmas condições de tráfego encontradas no *trace* CAIDA no ambiente do Mininet Hi-Fi representou uma dificuldade, uma vez que esse ambiente utiliza comutadores em software, o que de certa forma restringe seu desempenho.
- Nas avaliações do ODHIn com *traces* reais, tivemos uma dificuldade em obter altos valores de acurácia. Acreditamos que a modificação no algoritmo para preenchimento das tabelas *hash* pode ter gerado um efeito colateral, que precisa ser melhor investigado.

Durante o desenvolvimento deste trabalho foram identificados alguns direcionamentos para trabalhos futuros:

- Avaliação de outras estratégias de amostragem no Coletor, como a estratificada e a aleatória, poderia proporcionar melhores resultados mesmo utilizando baixas taxas de amostragem.
- A implementação de um mecanismo seletivo que possa aplicar um filtro sobre os pacotes que seriam espelhados para o Coletor. Dessa forma, ao invés de enviar uma cópia de todo o tráfego poderia-se enviar algumas faixa de tráfego com base nos prefixos da árvore que atingiram o limiar de detecção.
- O desenvolvimento de um mecanismo que possa limitar a quantidade de regras instaladas na tabela do comutador OpenFlow, uma vez que esse recurso é limitado nos atuais comutadores do mercado.
- O desenvolvimento de um mecanismo de adaptação que permita ao ODHIn se adaptar, caso as condições de tráfego mudem. Esse mecanismo poderia, por exemplo, monitorar a dispersão dos endereços IP observados.

Referências Bibliográficas

[1] AL-FARES, M.; RADHAKRISHNAN, S.; RAGHAVAN, B.; HUANG, N.; VAHDAT, A. **Hedera: Dynamic Flow Scheduling for Data Center Networks**. In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, p. 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[2] AREFIN, A.; SINGH, V.; JIANG, G.; ZHANG, Y.; LUMEZANU, C. **Diagnosing Data Center Behavior Flow by Flow**. In: *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, p. 11–20, July 2013.

[3] BALESTRA, G.; LUCIANO, S.; PIZZONIA, M.; VISSICCHIO, S. **Leveraging Router Programmability for Traffic Matrix Computation**. In: *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, p. 11:1–11:6, New York, NY, USA, 2010. ACM.

[4] BANDI, N.; METWALLY, A.; AGRAWAL, D.; EL ABBADI, A. **Fast Data Stream Algorithms using Associative Memories**. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, p. 247–256, 2007.

[5] BENSON, T.; AKELLA, A.; SHAIKH, A.; SAHU, S. **CloudNaaS: A Cloud Networking Platform for Enterprise Applications**. In: *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, p. 8:1–8:13, New York, NY, USA, 2011. ACM.

[6] BRAGA, R.; MOTA, E.; PASSITO, A. **Lightweight DDoS Flooding Attack Detection using NOX/OpenFlow**. In: *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, p. 408–415, Oct 2010.

[7] BRAUCKHOFF, D.; TELLENBACH, B.; WAGNER, A.; MAY, M.; LAKHINA, A. **Impact of Packet Sampling on Anomaly Detection Metrics**. In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, p. 159–164, New York, NY, USA, 2006. ACM.

[8] CACTI. **The Complete RRDTool-based Graphing Solution**. http://www.cacti.net/, 2014. [Última visita: 09-set-2014].

[9] CAIDA. **The CAIDA UCSD Anonymized Internet Traces 2013**. http://www.caida.org/data/passive/passive_2013_dataset.xml, 2013. [Última visita: 20-jul-2014].

[10] CASADO, M.; FREEDMAN, M. J.; PETTIT, J.; LUO, J.; MCKEOWN, N.; SHENKER, S. **Ethane: Taking Control of the Enterprise**. *SIGCOMM Computer Communication Review*, 37(4):1–12, Aug. 2007.

[11] CHINCHOR, N. **MUC-4 Evaluation Metrics**. In: *Conference on Message understanding*, p. 22–29, 1992.

[12] CHOWDHURY, S.; BARI, M.; AHMED, R.; BOUTABA, R. **PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks**. In: *Network Operations and Management Symposium (NOMS), 2014 IEEE*, p. 1–9, May 2014.

[13] CISCO. **NetFlow**. http://www.cisco.com/web/go/netflow, 2014. [Última visita: 10-set-2014].

[14] CORMODE, G.; HADJIELEFTHERIOU, M. **Finding Frequent Items in Data Streams**. *Proceedings of the VLDB Endowment*, 2008.

[15] CORMODE, G.; HADJIELEFTHERIOU, M. **Methods for Finding Frequent Items in Data Streams**. *The International Journal on Very Large Data Bases*, 2010.

[16] CORMODE, G.; KORN, F.; MUTHUKRISHNAN, S.; SRIVASTAVA, D. **Finding Hierarchical Heavy Hitters in Data Streams**. In: *International Conference on Very Large Data Bases*, p. 464–475, 2003.

[17] CORMODE, G.; KORN, F.; MUTHUKRISHNAN, S.; SRIVASTAVA, D. **Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-dimensional Data**. In: *ACM International Conference on Management of Data (SIGMOD)*, p. 155–166, 2004.

[18] CORMODE, G.; MUTHUKRISHNAN, S. **What's New: Finding Significant Differences in Network Data Streams**. *IEEE/ACM Transactions on Networking (TON)*, 13(6):1219–1232, Dec. 2005.

[19] CRISTIAN LUMEZANU - NEC LABS. **SDN/OpenFlow Reading List**. http://www.nec-labs.com/~lume/sdn-reading-list.html, 2014. [Última visita: 27-ago-2014].

[20] CURTIS, A. R.; MOGUL, J. C.; TOURRILHES, J.; YALAGANDULA, P.; SHARMA, P.; BANERJEE, S. **DevoFlow: Scaling Flow Management for High-performance Networks**. In: *Proceedings of the ACM SIGCOMM 2011 conference*, p. 254–265, 2011.

[21] DA CRUZ, M. A.; CASTRO E SILVA, L.; CORREA, S.; CARDOSO, K. V. **Accurate Online Detection of Bidimensional Hierarchical Heavy Hitters in Software-Defined Networks**. In: *Communications (LATINCOM), 2013 IEEE Latin-America Conference on*, p. 1–6, Nov 2013.

[22] DA CRUZ, M. A.; CARDOSO, K. V.; CORRÊA, S. L. **Detection of Bidimensional Hierarchical Heavy Hitters in OpenFlow Networks**. In: *IV Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF)*, 2013.

[23] DAS, A.; LUMEZANU, C.; ZHANG, Y.; SINGH, V.; JIANG, G.; YU, C. **Transparent and Flexible Network Management for Big Data Processing in the Cloud**. In: *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, Berkeley, CA, 2013. USENIX.

[24] ERICKSON, D.; GIBB, G.; HELLER, B.; HARA, M.; LANTZ, B.; PETTIT, J.; PFAFF, B.; SHERWOOD, R.; TALAYCO, D.; YABE, T.; YIAKOUMIS, Y. **OpenFlow Switch Specification Version 1.0.0**. http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf, Dec 2009. [Última visita: 29-ago-2014].

[25] ERICKSON, D.; GIBB, G.; HELLER, B.; HARA, M.; LANTZ, B.; PETTIT, J.; PFAFF, B.; SHERWOOD, R.; TALAYCO, D.; YABE, T.; YIAKOUMIS, Y. **OpenFlow Switch Specification Version 1.1.0**. http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf, Feb 2011. [Última visita: 03-set-2014].

[26] ESTAN, C.; SAVAGE, S.; VARGHESE, G. **Automatically Inferring Patterns of Resource Consumption in Network Traffic**. In: *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, p. 137–148, 2003.

[27] ESTAN, C.; VARGHESE, G. **New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice**. *ACM Trans. Comput. Syst.*, 21(3):270–313, Aug. 2003.

[28] FEAMSTER, N.; REXFORD, J.; ZEGURA, E. **The Road to SDN: An Intellectual History of Programmable Networks**. *SIGCOMM Computer Communication Review*, 44(2):87–98, Apr. 2014.

[29] FIBRE. **Future Internet testbeds/experimentation between BRazil and Europe**. http://www.fibre-ict.eu, 2014. [Última visita: 27-nov-2014].

[30] FOSTER, N.; GUHA, A.; REITBLATT, M.; STORY, A.; FREEDMAN, M.; KATTA, N.; MONSANTO, C.; REICH, J.; REXFORD, J.; SCHLESINGER, C.; WALKER, D.; HARRI-

SON, R. **Languages for Software-Defined Networks**. *Communications Magazine, IEEE*, 51(2):128–134, February 2013.

[31] FOSTER, N.; HARRISON, R.; FREEDMAN, M. J.; MONSANTO, C.; REXFORD, J.; STORY, A.; WALKER, D. **Frenetic: A Network Programming Language**. In: *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, p. 279–291, New York, NY, USA, 2011. ACM.

[32] GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. **NOX: Towards an Operating System for Networks**. *SIGCOMM Computer Communication Review*, 2008.

[33] GUPTA, M.; SOMMERS, J.; BARFORD, P. **Fast, Accurate Simulation for SDN Prototyping**. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, p. 31–36, New York, NY, USA, 2013. ACM.

[34] HANDIGOL, N.; FLAJSLIK, M.; SEETHARAMAN, S.; MCKEOWN, N.; JOHARI, R. **Aster* x: Load-balancing as a Network Primitive**. *9th GENI Engineering Conference (Plenary)*, p. 1–2, 2010.

[35] HANDIGOL, N.; HELLER, B.; JEYAKUMAR, V.; LANTZ, B.; MCKEOWN, N. **Reproducible Network Experiments using Container-based Emulation**. In: *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, p. 253–264, 2012.

[36] HELLER, B.; SEETHARAMAN, S.; MAHADEVAN, P.; YIAKOUMIS, Y.; SHARMA, P.; BANERJEE, S.; MCKEOWN, N. **ElasticTree: Saving Energy in Data Center Networks**. In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, p. 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[37] HERNANDEZ, E. A.; CHIDESTER, M. C.; GEORGE, A. D. **Adaptive Sampling for Network Management**. *Journal of Network and Systems Management, Springer*, 9(4):409–434, Dec. 2001.

[38] HOFSTEDE, R.; CELEDA, P.; TRAMMELL, B.; DRAGO, I.; SADRE, R.; SPEROTTO, A.; PRAS, A. **Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX**. *Communications Surveys Tutorials, IEEE*, PP(99):1–1, 2014.

[39] HU, C.; WANG, S.; TIAN, J.; LIU, B.; CHENG, Y.; CHEN, Y. **Accurate and Efficient Traffic Monitoring Using Adaptive Non-Linear Sampling Method**. In: *INFOCOM*

2008. The 27th Conference on Computer Communications. IEEE, p. 26–30, April 2008.

[40] HU, F.; HAO, Q.; BAO, K. **A Survey on Software Defined Networking (SDN) and OpenFlow: From Concept to Implementation**. *Communications Surveys Tutorials, IEEE*, PP(99):1–1, 2014.

[41] IETF. **Packet Sampling (PSAMP) Working Group**. http://datatracker.ietf.org/wg/psamp/charter/, 2009. [Última visita: 23-ago-2014].

[42] INTERNET ENGINEERING TASK FORCE (IETF). **RFC6437: IPv6 Flow Label Specification**. http://tools.ietf.org/html/rfc6437, 2011. [Última visita: 14-nov-2014].

[43] INTERNET ENGINEERING TASK FORCE (IETF). **RFC7011: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information**. http://tools.ietf.org/html/rfc7011, 2013. [Última visita: 13-ago-2014].

[44] INTERNET ENGINEERING TASK FORCE (IETF). **Forwarding and Control Element Separation Working Group**. http://datatracker.ietf.org/wg/forces/charter/, 2014. [Última visita: 20-out-2014].

[45] INTERNET ENGINEERING TASK FORCE (IETF). **IP Flow Information Export Working Group**. http://datatracker.ietf.org/wg/IPFIX/charter/, 2014. [Última visita: 13-ago-2014].

[46] ISHIMORI, A.; FARIAS, F.; CERQUEIRA, E.; ABELEM, A. **Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking**. In: *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, p. 81–86, Oct 2013.

[47] JAIN, S.; KUMAR, A.; MANDAL, S.; ONG, J.; POUTIEVSKI, L.; SINGH, A.; VENKATA, S.; WANDERER, J.; ZHOU, J.; ZHU, M.; ZOLLA, J.; HÖLZLE, U.; STUART, S.; VAHDAT, A. **B4: Experience with a Globally-deployed Software Defined Wan**. In: *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, p. 3–14, 2013.

[48] JAMES LIAO - PICA8 OPEN NETWORKING. **SDN System Performance**. http://pica8.org/blogs/?p=201, June 2012. [Última visita: 29-ago-2014].

[49] JOSE, L.; YU, M.; REXFORD, J. **Online Measurement of Large Traffic Aggregates on Commodity Switches**. In: *USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, p. 13–13, 2011.

[50] JURGA, R. E.; HULBÓJ, M. M. **Packet Sampling for Network Monitoring**. Technical report, CERN - HP Procurve openlab project, Dec 2007.

[51] KIM, H.; FEAMSTER, N. **Improving Network Management with Software Defined Networking**. *Communications Magazine, IEEE*, 51(2):114–119, February 2013.

[52] KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. **Software-Defined Networking: A Comprehensive Survey**. *Computing Research Repository, ArXiv e-prints*, abs/1406.0440, 2014.

[53] LANTZ, B.; HELLER, B.; MCKEOWN, N. **A Network in a Laptop: Rapid Prototyping for Software-Defined Networks**. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, p. 19:1–19:6, New York, NY, USA, 2010. ACM.

[54] LARA, A.; KOLASANI, A.; RAMAMURTHY, B. **Network Innovation using OpenFlow: A Survey**. *Communications Surveys Tutorials, IEEE*, 16(1):493–512, First 2014.

[55] MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. **OpenFlow: Enabling Innovation in Campus Networks**. *ACM SIGCOMM Computer Communication Review*, 2008.

[56] METWALLY, A.; AGRAWAL, D.; EL ABBADI, A. **Efficient Computation of Frequent and Top-k Elements in Data Streams**. In: *Proceedings of the 10th international conference on Database Theory*, p. 398–412, 2005.

[57] MITZENMACHER, M.; STEINKE, T.; THALER, J. **Hierarchical Heavy Hitters with the Space Saving Algorithm**. In: *Proceedings of the Meeting on Algorithm Engineering and Experiments*, ALENEX '12, p. 160–174, 2012.

[58] MOGUL, J. C.; CONGDON, P. **Hey, You Darned Counters!: Get off my ASIC!** In: *Proceedings of the first workshop on Hot topics in software defined networks*, p. 25–30, 2012.

[59] MOSHREF, M.; YU, M.; GOVINDAN, R. **Resource/Accuracy Tradeoffs in Software-defined Measurement**. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, p. 73–78, New York, NY, USA, 2013. ACM.

[60] NASCIMENTO, M. R.; ROTHENBERG, C. E.; SALVADOR, M. R.; CORRÊA, C. N. A.; DE LUCENA, S. C.; MAGALHÃES, M. F. **Virtual Routers As a Service: The RouteFlow Approach Leveraging Software-Defined Networks**. In: *Proceedings*

of the 6th International Conference on Future Internet Technologies, CFI '11, p. 34–37, New York, NY, USA, 2011. ACM.

[61] NETFPGA. **NetFPGA Technical Specifications**. http://netfpga.org/1G_specs.html, 2013. [Última visita: 23-mar-2013].

[62] NETWORK WORKING GROUP (IETF). **RFC1157: A Simple Network Management Protocol (SNMP)**. http://tools.ietf.org/html/rfc1157, 1990. [Última visita: 23-set-2014].

[63] NETWORK WORKING GROUP (IETF). **RFC2722: Traffic Flow Measurement: Architecture**. http://tools.ietf.org/html/rfc2722, 1999. [Última visita: 14-nov-2014].

[64] NETWORK WORKING GROUP (IETF). **RFC3917: Requirements for IP Flow Information Export (IPFIX)**. http://tools.ietf.org/html/rfc3917, 2004. [Última visita: 14-nov-2014].

[65] NUNES, B.; MENDONCA, M.; NGUYEN, X.; OBRACZKA, K.; TURLETTI, T. **A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks**. *Communications Surveys Tutorials, IEEE*, PP(99):1–18, 2014.

[66] ONF. **Member Listing - Open Networking Foundation**. https://www.opennetworking.org/membership/member-listing, 2014. [Última visita: 16-ago-2014].

[67] OPEN NETWORKING FOUNDATION. **Software-Defined Networking (SDN) Definition**. https://www.opennetworking.org/sdn-resources/sdn-definition, 2014. [Última visita: 22-jul-2014].

[68] OPEN NETWORKING FOUNDATION (ONF). **OpenFlow Switch Specification Version 1.2**. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf, Dec 2011. [Última visita: 03-set-2014].

[69] OPEN NETWORKING FOUNDATION (ONF). **OpenFlow Switch Specification Version 1.3.0**. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf, Apr 2012. [Última visita: 03-set-2014].

[70] OPEN NETWORKING FOUNDATION (ONF). **OpenFlow Switch Specification Version 1.4.0**. https://www.opennetworking.org/images/

stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf, Oct 2013. [Última visita: 03-set-2014].

[71] PACKET SAMPLING WORKING GROUP (IETF). **RFC5474: A Framework for Packet Selection and Reporting**. http://tools.ietf.org/html/rfc5474, 2009. [Última visita: 23-ago-2014].

[72] PACKET SAMPLING WORKING GROUP (IETF). **RFC5475: Sampling and Filtering Techniques for IP Packet Selection (Proposed Standard)**. http://tools.ietf.org/html/rfc5475, 2009. [Última visita: 13-ago-2014].

[73] PACKET SAMPLING WORKING GROUP (IETF). **RFC5476: Packet Sampling (PSAMP) Protocol Specifications (Proposed Standard)**. http://tools.ietf.org/html/rfc5476, 2009. [Última visita: 23-ago-2014].

[74] PACKET SAMPLING WORKING GROUP (IETF). **RFC5477: Information Model for Packet Sampling Exports (Proposed Standard)**. http://tools.ietf.org/html/rfc5477, 2009. [Última visita: 23-ago-2014].

[75] PESCAPE, A.; ROSSI, D.; TAMMARO, D.; VALENTI, S. **On the Impact of Sampling on Traffic Monitoring and Analysis**. In: *Teletraffic Congress (ITC), 2010 22nd International*, p. 1–8, Sept 2010.

[76] PORRAS, P.; SHIN, S.; YEGNESWARAN, V.; FONG, M.; TYSON, M.; GU, G. **A Security Enforcement Kernel for OpenFlow Networks**. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, p. 121–126, New York, NY, USA, 2012. ACM.

[77] POX. **POX**. http://www.noxrepo.org/pox/about-pox/, 2013. [Última visita: 23-mar-2013].

[78] QAZI, Z. A.; TU, C.-C.; CHIANG, L.; MIAO, R.; SEKAR, V.; YU, M. **SIMPLE-fying Middlebox Policy Enforcement Using SDN**. In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, p. 27–38, New York, NY, USA, 2013. ACM.

[79] REITBLATT, M.; CANINI, M.; GUHA, A.; FOSTER, N. **FatTire: Declarative Fault Tolerance for Software-Defined Networks**. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, p. 109–114, New York, NY, USA, 2013. ACM.

[80] RIJSBERGEN, C. J. V. **Information Retrieval**. Butterworth-Heinemann, 1979.

[81] ROTHENBERG, C. E.; NASCIMENTO, M. R.; SALVADOR, M. R.; CORRÊA, C. N. A.; CUNHA DE LUCENA, S.; RASZUK, R. **Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking**. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, p. 13–18, New York, NY, USA, 2012. ACM.

[82] SHIN, S.; PORRAS, P. A.; YEGNESWARAN, V.; FONG, M. W.; GU, G.; TYSON, M. **FRESCO: Modular Composable Security Services for Software-Defined Networks**. In: *Internet Society NDSS*, Feb 2013.

[83] SILVA, J. M. C.; CARVALHO, P.; LIMA, S. R. **Computational Weight of Network Traffic Sampling Techniques**. In: *Computers and Communications (ISCC), 2014 IEEE Symposium on*, June 2014.

[84] SPLUNK. **Operational Intelligence, Log Management Solution**. http://www.splunk.com/, 2014. [Última visita: 09-set-2014].

[85] SURESH, L.; SCHULZ-ZANDER, J.; MERZ, R.; FELDMANN, A.; VAZAO, T. **Towards Programmable Enterprise WLANS with Odin**. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, p. 115–120, New York, NY, USA, 2012. ACM.

[86] TAMMARO, D.; VALENTI, S.; ROSSI, D.; PESCAPÉ, A. **Exploiting Packet-Sampling Measurements for Traffic Characterization and Classification**. *International Journal of Network Management, Wiley*, 22(6):451–476, 2012.

[87] TOOTOONCHIAN, A.; GHOBADI, M.; GANJALI, Y. **OpenTM: Traffic Matrix Estimator for OpenFlow Networks**. In: *Proceedings of the 11th International Conference on Passive and Active Measurement*, PAM'10, p. 201–210, Berlin, Heidelberg, 2010. Springer-Verlag.

[88] VAN ADRICHEM, N.; DOERR, C.; KUIPERS, F. **OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks**. In: *Network Operations and Management Symposium (NOMS), 2014 IEEE*, p. 1–8, May 2014.

[89] VOELLMY, A.; KIM, H.; FEAMSTER, N. **Procera: A Language for High-level Reactive Network Control**. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, p. 43–48, New York, NY, USA, 2012. ACM.

[90] VOELLMY, A.; WANG, J. **Scalable Software Defined Network Controllers**. *SIGCOMM Computer Communication Review, ACM*, 42(4):289–290, Aug. 2012.

[91] WANG, S.-Y.; CHOU, C.-L.; YANG, C.-M. **EstiNet Openflow Network Simulator and Emulator**. *Communications Magazine, IEEE*, 51(9):110–117, September 2013.

[92] YAN, Z.; TRACY, C.; VEERARAGHAVAN, M. **A Hybrid Network Traffic Engineering System**. In: *High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on*, p. 141–146, June 2012.

[93] YAP, K.-K.; KOBAYASHI, M.; SHERWOOD, R.; HUANG, T.-Y.; CHAN, M.; HANDIGOL, N.; MCKEOWN, N. **OpenRoads: Empowering Research in Mobile Networks**. *SIGCOMM Computer Communication Review, ACM*, 40(1):125–126, Jan. 2010.

[94] YEGANEH, S.; TOOTOONCHIAN, A.; GANJALI, Y. **On Scalability of Software-Defined Networking**. *Communications Magazine, IEEE*, 51(2):136–141, February 2013.

[95] YU, C.; LUMEZANU, C.; ZHANG, Y.; SINGH, V.; JIANG, G.; MADHYASTHA, H. V. **FlowSense: Monitoring Network Utilization with Zero Measurement Cost**. In: *Proceedings of the 14th International Conference on Passive and Active Measurement*, PAM'13, p. 31–41, Berlin, Heidelberg, 2013. Springer-Verlag.

[96] YU, M.; JOSE, L.; MIAO, R. **Software Defined Traffic Measurement with OpenSketch**. In: *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, p. 29–42, 2013.

[97] ZABBIX. **An Enterprise-Class Open Source Distributed Monitoring Solution**. http://www.zabbix.com/, 2014. [Última visita: 09-set-2014].

[98] ZHANG, Y.; BRESLAU, L.; PAXSON, V.; SHENKER, S. **On the Characteristics and Origins of Internet Flow Rates**. *SIGCOMM Computer Communication Review*, 32(4):309–322, Aug. 2002.

[99] ZHANG, Y.; SINGH, S.; SEN, S.; DUFFIELD, N.; LUND, C. **Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Applications**. In: *ACM Internet Measurement Conference (IMC)*, p. 101–114, 2004.

[100] ZHAO, Q.; XU, J.; LIU, Z. **Design of a Novel Statistics Counter Architecture with Optimal Space and Time Efficiency**. In: *Proceedings of the joint international conference on Measurement and modeling of computer systems*, p. 323–334, 2006.

APÊNDICE A

Pseudo-código do algoritmo $hhh1D$

Algoritmo A.1: Pseudo-código *hhh*1*D*

entrada: ϕ: limiar de detecção de HHH; *S*: conjunto de pacotes
saída : *H*: conjunto de HHHs encontrados; *R*: conjunto de regras OpenFlow

```
1  arvore ← criaArvoreInicial();
2  totalContadores ← 0;
3  listaMon ← [];
4  proximaListaMon ← [];
5  listaMon ← recuperaNos(arvore);
6  proximaListaMon ← expandeNos(listaMon);
7  para cada pacote em S faça
8      elem ← encontraCombinacao(pacote, proximaListaMon);
9      proximaListaMon[elem] ← proximaListaMon[elem] + 1;
10     totalContadores ← totalContadores + 1;
11 fim
12 limiar ← calculaLimiar(φ, totalContadores);
13 novaListaMon ← [];
14 para cada item em listaMon faça
15     se elementoRaiz(item) então
16         adicionaNo(item, novaListaMon);
17     senão se (item[contador] > limiar) então
18         para cada filho em [filhoDireita(item), filhoEsquerda(item)] faça
19             se (filho[contador] > limiar) então
20                 adicionaNo(filho, novaListaMon);
21             senão
22                 contadorFilho ← filho[contador];
23                 noPai ← recuperaPai(filho);
24                 noPai[contador] ← noPai[contador] + contadorFilho;
25             fim
26         fim
27         se ([filhoDireita(item), filhoEsquerda(item)] nao estao em novaListaMon)
           então
28             adicionaNo(item, novaListaMon);
29         fim
30     senão
31         contadorNo ← item[contador];
32         noPai ← recuperaPai(item);
33         noPai[contador] ← noPai[contador] + contadorNo;
34     fim
35 fim
36 proximaListaMon ← expandeNos(novaListaMon);
37 listaMon ← novaListaMon;
38 H ← listaMon;
39 R ← proximaListaMon;
40 reporta H, R;
```