Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Monitoramento de Desempenho com Middleboxes em Redes Definidas por Software

Ethel Barreto Gondim

Dissertação apresentada como requisito parcial
para conclusão do Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Divanilson Rodrigo de Sousa Campelo

Coorientador
Prof. Dr. André Costa Drummond

Brasília
2015

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

Gondim, Ethel Barreto
GG637m Monitoramento de Desempenho com Middleboxes em Redes Definidas por Software / Ethel Barreto Gondim; orientador Divanilson Rodrigo de Sousa Campelo; co orientador André Costa Drummond. -- Brasília, 2015.
77 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2015.

1. Gerência de Redes. 2. Gerência de Desempenho. 3. Gerência de Desempenho de Aplicações. 4. Redes Definidas por Software - SDN. 5. Middlebox. I. Campelo, Divanilson Rodrigo de Sousa, orient. II. Drummond, André Costa, co-orient. III. Título.

**Universidade de Brasília**
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Monitoramento de Desempenho com Middleboxes em Redes Definidas por Software

Ethel Barreto Gondim

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Divanilson Rodrigo de Sousa Campelo (Orientador)
Centro de Informática/UFPE

Prof.ª Dr.ª Priscila América Solis Mendez Barreto
Departamento de Ciência da Computação/UnB

Prof. Dr. Rafael Timóteo de Sousa Júnior
Departamento de Engenharia Elétrica/UnB

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 07 de agosto de 2015

# Agradecimentos

Agradeço à minha família, que me apoiou, aconselhou e incentivou em todos os momentos da minha vida. Aos meus amigos e namorado, pela cumplicidade, compreensão, palavras de apoio e pelos bons momentos que compartilhamos. Ao professor e orientador Divanilson, pela inspiração, compreensão e incentivo constantes desde a minha graduação. Ao colega Janael, pela ajuda no desenvolvimento de trabalhos conjuntos durante o Mestrado. Aos demais professores da Universidade de Brasília, pela formação de uma base sólida para o desenvolvimento deste trabalho.

# Resumo

O gerenciamento de desempenho de aplicações é frequentemente dificultado pela presença de middleboxes, por sua variedade e capacidade de alterar o tráfego que os atravessa. Com o advento das Redes Definidas por Software (do Inglês, *Software-Defined Networking*, SDN), surgem novas possibilidades para o gerenciamento de desempenho a partir da programabilidade dos dispositivos e do controle centralizado do tráfego. Este trabalho propõe uma arquitetura que objetiva mitigar os desafios impostos pelos middleboxes ao monitoramento de desempenho em SDN. Em particular, é apresentado e validado um protótipo que identifica o tempo de resposta, a disponibilidade e informações de conexões de aplicações na presença de quatro middleboxes: um balanceador de carga, um *firewall*, um sistema de prevenção de intrusões (do Inglês, *Intrusion Prevention System*, IPS) e um sistema de tradução de endereços de rede (do Inglês, *Network Address Translation*, NAT). Para os três primeiros middleboxes, foram desenvolvidas Interfaces de Programação de Aplicações (do Inglês, *Application Programming Interfaces*, APIs) específicas.

**Palavras-chave:** Gerência de Redes, Gerência de Desempenho, Gerência de Desempenho de Aplicações, Redes Definidas por Software - SDN, Middlebox

# Abstract

Application Performance management is frequently hampered by the presence of middleboxes, because of their variety and capacity of modifying the traffic that traverses them. With the advent of Software-Defined Networking (SDN), new possibilities for performance management arise from the programmability of devices and the centralized control of traffic. This work proposes an architecture that aims at mitigating the challenges posed by middleboxes in performance monitoring in SDN. In particular, it is presented and validated a prototype that identifies the response time, the availability and connection information of applications in the presence of four middleboxes: a load balancer, a firewall, an Intrusion Prevention System (IPS) and a Network Address Translation (NAT) system. For the first three middleboxes, specific Application Programming Interfaces (APIs) were developed.

**Keywords:** Network Management, Performance Management, Application Performance Management, Software-Defined Networking - SDN, Middlebox

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

O Gerenciamento de Desempenho de Aplicações (do Inglês, *Application Performance Management*, APM) tem por objetivo prover visibilidade do comportamento dos sistemas críticos ao funcionamento de uma organização. Essa visibilidade muitas vezes é dificultada pela presença de middleboxes, dispositivos essenciais às redes corporativas. Esses dispositivos podem modificar o tráfego de formas variadas, e comumente há pouco controle sobre essas modificações. No contexto de Redes Definidas por Software (do Inglês, *Software-Defined Networking*, SDN), a estratégia de controle da rede é profundamente modificada, trazendo novos desafios e oportunidades para o gerenciamento tanto da infraestrutura de rede como das aplicações executadas sobre ela. Este capítulo contextualiza os desafios do gerenciamento de desempenho em SDN e na presença de middleboxes, e introduz os objetivos e a proposta deste trabalho.

## 1.1 Contextualização

As soluções de gerenciamento de desempenho de aplicações têm por objetivo prover informações úteis para a melhoria ou manutenção do desempenho de sistemas críticos, em geral com foco na avaliação da experiência dos usuários finais desses sistemas. Elas auxiliam a equipe de Tecnologia da Informação (TI) na manutenção de altos níveis de disponibilidade, no atendimento de acordos de níveis de serviço e na rápida identificação e solução de problemas, além de possibilitarem maior eficiência na utilização e planejamento dos recursos de infraestrutura [36]. Essas soluções podem trazer uma vasta gama de informações, como métricas momentâneas de um conjunto de transações ou conexões, medições históricas do comportamento de componentes, análises de tendências para planejamento de capacidade, alertas de mau funcionamento, dentre outros.

Tais soluções têm uma presença crescente no mercado, o que evidencia uma dependência cada vez maior dos negócios em relação a sistemas computacionais. Segundo Will Cappelli (Gartner), em tradução livre, "os processos de negócios das empresas estão tão digitalizados que já não há distinção entre os negócios e a TI" [22]. É cada vez mais claro que garantir continuamente a disponibilidade e o desempenho das aplicações de uma organização é essencial, visto que essas características têm grande impacto em seus objetivos de negócio. À medida que as aplicações se tornam mais complexas e distribuídas, e que as infraestruturas se tornam mais híbridas e dinâmicas, as ferramentas de APM se tornam cada vez mais essenciais à administração do ambiente de TI.

Ainda que as ferramentas de APM tenham evoluído consideravelmente nos últimos anos, a percepção que elas conseguem obter do comportamento das aplicações é frequentemente prejudicada ou dificultada pela presença de middleboxes. Definidos como dispositivos físicos ou virtuais que atuam sobre o tráfego com propósitos que vão além do roteamento/encaminhamento tradicional de pacotes, os middleboxes têm uma presença marcante na Internet atualmente [9], e frequentemente superam a quantidade de roteadores comuns em redes corporativas [35], conforme indicado na Figura 1.1 e na Tabela 1.1. Nessa categoria, enquadram-se NATs, *firewalls* IP e de aplicação, *gateways* de aplicação, *proxies*, *caches*, sistemas de detecção/prevenção de intrusão e balanceadores de carga, dentre outros.

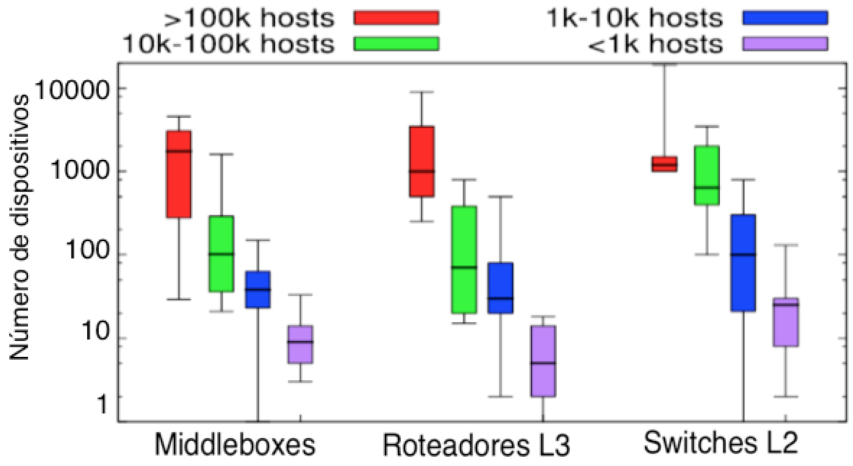



Figura 1.1: Comparação entre a quantidade de roteadores, *switches* de camada 2 e middleboxes em uma pesquisa realizada com 57 operadores de redes (imagem adaptada de [35]).

Tabela 1.1: Quantidades de roteadores e de diversos tipos de middleboxes em uma rede corporativa, em que o total de middleboxes é cerca de 70% do total de roteadores (dados retirados de [35]).

| **Tipo de Middlebox** | **Quantidade** |
|---|---|
| *Firewalls* | 166 |
| IDS/IPS | 127 |
| *Gateways* de Mídia | 110 |
| Balanceadores de Carga | 67 |
| *Proxies* | 66 |
| *Gateways* VPN | 45 |
| Otimizadores WAN | 44 |
| *Gateways* de Voz | 11 |
| **Total Middleboxes** | 636 |
| **Total Roteadores** | 900 |

Por serem muito variados e existirem em grande quantidade, os middleboxes adicionam complexidade significativa ao processo de monitoramento e gerenciamento. Há uma

complexidade inerente ao entendimento do comportamento de cada um dos middleboxes, o que dificulta o trabalho das soluções de gerenciamento na busca de informações reais e condizentes com o comportamento dos ambientes avaliados. Além da diversidade de funcionalidades, comportamentos e fabricantes, os middleboxes podem ser implementados em localizações arbitrárias e nas mais variadas formas, como *appliances* físicos/dedicados, máquinas virtuais, coleções de processos, funcionalidade em um controlador SDN, dentre outros [25].

Como exemplo, pode-se citar o caso do acompanhamento de transações de uma aplicação que contém um balanceador de carga direcionando o tráfego para um *pool* de servidores. Uma abordagem de gerenciamento razoavelmente comum seria observar essas transações a partir do tráfego de rede. Nesse caso, as transações seriam entendidas como "terminadas" na chegada ao balanceador, pois a comunicação tomaria sequência com um IP de origem diferente (o IP virtual do balanceador) para chegar ao servidor de destino no *pool*. Assim, a solução de gerenciamento não acompanharia a transação completa sem compreender o comportamento do balanceador. Um raciocínio semelhante pode ser estendido para qualquer middlebox que provoque uma alteração no tráfego que não seja "prevista" pela solução de gerenciamento. A tarefa de "prever" ou "interpretar" a atuação de todos os middleboxes disponíveis, no entanto, é extremamente complexa para soluções de gerenciamento tradicionais, por toda a variedade desses dispositivos, conforme comentado acima.

A grande diversidade e complexidade de dispositivos presentes na rede foram parte da motivação para o surgimento das Redes Definidas por Software [19]. Em SDN, propõe-se a separação do plano de controle do plano de encaminhamento dos dados, conforme ilustrado na Figura 1.2. A rede passa a ter um plano de controle gerenciado de forma mais centralizada, modificando-se o paradigma de funcionamento dos ativos de redes tradicionais [44]. Esse gerenciamento centralizado se dá em um ou mais controladores de rede, que recebem os registros dos diversos fluxos de tráfego que passam pela rede. Com essa centralização, surgem novas oportunidades para ferramentas de monitoramento, que podem simplificar o processo de obtenção de dados por elas. Além disso, a arquitetura de SDN propõe uma maior programabilidade dos dispositivos presentes nas redes. Isso implica a existência de um padrão de comunicação com os dispositivos, como o OpenFlow [5], e também uma maior presença de Interfaces de Programação de Aplicações (do Inglês, *Application Programming Interfaces*, APIs). Essas APIs podem ser utilizadas para obter/enviar dados a virtualmente qualquer dispositivo, inclusive os middleboxes.

O tema de SDN tem recebido crescente atenção do mercado e da comunidade acadêmica nos últimos anos, sendo notáveis as iniciativas de grupos de pesquisa renomados, como o da Universidade de Stanford [49], e de grandes empresas de TI, como Google [27] e VMware [54]. A abordagem proposta nessas redes já vem sendo utilizada em diversas organizações, com ferramentas como o VMware NSX [54], e com soluções abertas baseadas no padrão OpenFlow, principalmente. No entanto, ainda não é notável a presença de soluções de APM que tomem proveito das características de SDN para simplificar e otimizar o processo de monitoramento de desempenho.

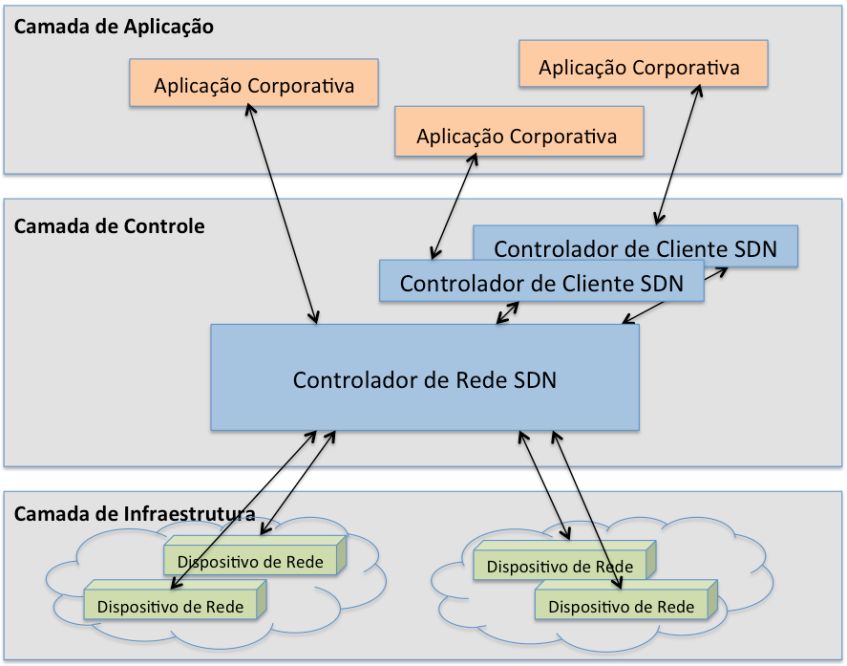



Figura 1.2: Arquitetura de uma rede SDN (adaptado de [56]).

## 1.2 Desafios e Oportunidades

Há expressivos desafios envolvendo os temas de middleboxes em SDN e de gerenciamento de desempenho desses middleboxes em geral. Por outro lado, as características de SDN trazem também uma série de novas oportunidades para a simplificação dos processos de gerenciamento. Alguns dos principais desafios e oportunidades identificados envolvendo os temas de middleboxes, gerenciamento e SDN são listados a seguir:

1. *Como aproveitar características de SDN para melhorar o monitoramento/gerenciamento?*
   A programabilidade dos dispositivos e a centralização dos dados de rede em controladores, ambos propostos na arquitetura de SDN, têm o potencial de modificar substancialmente o monitoramento e o gerenciamento tradicionais. Eles trazem novas possibilidades na implementação dinâmica de políticas e na simplificação dos mecanismos de controle.

2. *No que o controle centralizado dos controladores SDN pode facilitar o acompanhamento de transações?*
   É intuitivo concluir que o monitoramento de transações seria muito mais simples caso as informações de todos os fluxos de uma rede estivessem disponíveis em poucos pontos centralizados, o que é o caso dos controladores de rede em SDN.

3. *Como gerenciar o estado de middleboxes em SDN de forma unificada?*
   O gerenciamento do estado dos middleboxes é essencial para o controle de suas políticas e para o monitoramento de desempenho, mas pode se tornar complexo devido

à sua enorme diversidade estrutural e semântica. Outra dificuldade é que informações de comportamento e estado dos middleboxes muitas vezes não são acessíveis a qualquer tipo de controle externo. Além disso, existem diversos casos em que a atuação de um middlebox é associada a um conjunto de fluxos desconhecido, dificultando a aplicação de políticas em um formato "por fluxo", como é realizado em controladores de rede em SDN [25].

4. *Como adaptar middleboxes à dinamicidade de infraestrutura?*
   Com a presença cresente de infraestruturas heterogêneas e de implementação dinâmica, torna-se cada vez mais necessário automatizar a adaptação de middleboxes a novos cenários.

Este trabalho aborda os três primeiros desafios enumerados acima, com a intenção de tratar aspectos relativos apenas ao monitoramento de desempenho com middleboxes em SDN. A arquitetura proposta, detalhada no Capítulo 3, utiliza características de SDN para simplificar o monitoramento de aplicações em redes com middleboxes.

## 1.3 Objetivos

Este trabalho objetiva trazer maior simplicidade e eficiência à tarefa de gerenciamento de desempenho de aplicações em SDN, com ênfase em situações em que há um ou mais middleboxes interferindo no tráfego das aplicações. Há, portanto, dois segmentos da área de gerenciamento de desempenho que aqui são abordados conjuntamente:

- *O monitoramento/gerenciamento voltado para Redes Definidas por Software*: devido a SDN ser um paradigma relativamente recente e que trata especificamente das camadas de encaminhamento e roteamento, o desenvolvimento de soluções que trabalhem no gerenciamento da camada de aplicação para essas redes não é vasto. Nota-se que a grande maioria das soluções de APM de mercado não estão adaptadas para o funcionamento em SDN, e caso estejam, ainda trabalham de forma "tradicional" (ou seja, sem se utilizar de características de SDN para simplificar e otimizar o processo de obtenção de informações de monitoramento). Nesse contexto, este trabalho busca propor um mecanismo de monitoramento eficiente e simplificado para SDN, tomando proveito da centralização de fluxos nos controladores de redes;
- *O gerenciamento com middleboxes*: a presença de middleboxes, ainda que essencial, traz uma série de dificuldades ao monitoramento tradicional, devido à complexidade em se interagir com dispositivos extremamente variados em termos de funcionalidades, fabricantes e implementação. Este trabalho objetiva trazer também uma nova proposta para contornar as limitações que a presença de middleboxes impõe ao gerenciamento de desempenho, a partir da programabilidade dos dispositivos de rede em SDN.

Nesse contexto, o presente trabalho procura propor e validar uma arquitetura de solução de gerenciamento de desempenho de aplicações para SDN. Dessa forma, seus objetivos específicos são:

- A elaboração de uma arquitetura de gerenciamento de desempenho que colete fluxos de forma centralizada no(s) controlador(es) da rede e que verifique continuamente os estados de middleboxes;
- O desenvolvimento de um protótipo de solução de APM capaz de identificar o tempo de resposta e a disponibilidade de aplicações em uma rede SDN na presença de middleboxes, bem como algumas informações de rede das conexões das aplicações, utilizando a arquitetura proposta;
- A validação do protótipo como uma base apropriada para compor uma solução de APM completa e aplicável a ambientes complexos.

## 1.4 Estrutura do Trabalho

O texto a seguir se subdivide da seguinte forma:

**Capítulo 2:**

Apresenta os conceitos teóricos necessários ao entendimento deste trabalho, as principais ferramentas utilizadas em sua implementação, e a revisão do estado da arte realizada para a composição deste trabalho.

**Capítulo 3:**

Descreve conceitualmente a solução proposta e a metodologia utilizada neste trabalho.

**Capítulo 4:**

Detalha a implementação do protótipo de solução de gerenciamento de desempenho para redes SDN com middleboxes.

**Capítulo 5:**

Apresenta as etapas de validação do protótipo, descrevendo as avaliações realizadas e seus resultados.

**Capítulo 6:**

Conclui o trabalho, descrevendo suas contribuições e propondo trabalhos futuros.

# Capítulo 2

# Fundamentação Teórica

O presente capítulo introduz alguns conceitos básicos teóricos tratados neste trabalho, e apresenta as principais ferramentas utilizadas para seu desenvolvimento e a revisão do estado da arte do tema de pesquisa abordado.

## 2.1 Redes Definidas por Software

As Redes Definidas por Software são uma abordagem arquitetônica que simplifica as operações de rede permitindo maior interação entre aplicações/serviços e dispositivos de rede, sejam eles reais ou virtualizados [33]. Isso é realizado a partir da presença de um ou mais pontos de controle da rede logicamente centralizados, geralmente chamados de controladores SDN, que organizam, fazem a mediação e facilitam a comunicação entre os elementos da rede e as aplicações que desejam interagir com eles. Cada controlador então provê funções e operações de rede via interfaces amigáveis, programáticas e bidirecionais para aplicações modernas.

A existência de controladores explicita uma característica fundamental das Redes Definidas por Software, que é a separação entre o plano de controle e o plano de encaminhamento dos dados. Ou seja, a lógica de tomada de decisão é em grande parte ou totalmente movida dos dispositivos de rede para o(s) controlador(es). Essa separação viabiliza a composição da rede com equipamentos de propósito geral (*commodities*), mais simples e baratos, sendo suas funcionalidades realizadas por *softwares* especializados [31]. A implementação dessas funcionalidades depende da padronização de APIs independentes do fabricante do equipamento [44]. Essa forma de implementação das redes SDN viabiliza a programação do comportamento dos dispositivos de rede e *middlewares*, seja pelos fabricantes de equipamentos ou pelos próprios administradores de uma rede. Assim, as Redes Definidas por Software são um paradigma inovador de arquitetura e controle, extremamente interessante para o barateamento, a escalabilidade e a flexibilidade de redes de computadores em geral.

A programabilidade e a presença de APIs em SDN são comentadas a seguir. Discutem-se também alguns dos padrões e *frameworks* disponíveis para sua a implementação, e algumas das ferramentas voltadas para SDN utilizadas neste trabalho.

### 2.1.1 Programabilidade e Interfaces

O paradigma de SDN prevê a programabilidade dos serviços e aplicações que são executados sobre a infraestrutura de rede, dada a existência de um plano de controle centralizado em um ou mais controladores de rede. Essa programabilidade se dá por meio de APIs, e dá suporte a inovações em aplicações e protocolos. As APIs são classificadas como *northbound* ou *southbound*, como é descrito a seguir.

#### APIs *Northbound*

As APIs *northbound* são interfaces de comunicação entre o controlador de uma rede e os serviços e aplicações executados sobre ela, como ilustrado na Figura 2.1. Sua função é prover uma abstração em "alto nível" do funcionamento interno da rede para os desenvolvedores de aplicação, simplificando e possivelmente tornando mais dinâmico o desenvolvimento das aplicações. Dentre as utilidades possíveis para essas interfaces, estão a integração com pilhas de automação (como Puppet, Chef, SaltStack) e com plataformas de orquestração (como OpenStack e VMware vCloudDirector), e a otimização de aplicações de rede, em que se enquadram middleboxes como balanceadores de carga, *firewalls* e sistemas de detecção/prevenção de intrusões. Devido à enorme variedade de aplicações existentes, há também uma grande diversidade de APIs *northbound* disponíveis em diferentes níveis das pilhas de controle. Segundo [12], a consolidação dessas APIs ainda é um trabalho em andamento, e o seu desenvolvimento hoje é um dos maiores focos da *Open Networking Foundation* (ONF)[1].

#### APIs *Southbound*

As APIs *southbound* são interfaces de comunicação entre um controlador de rede SDN e os *switches* e roteadores, conforme ilustrado na Figura 2.1. Elas provêem o controle centralizado sobre os dispositivos de rede, viabilizando mudanças dinâmicas a partir de um controlador, e possibilitando a adaptação de suas configurações de acordo com demandas em tempo real. Existem diversos padrões de APIs *southbound* abertos e proprietários. O primeiro e mais comum dos padrões abertos é o OpenFlow, detalhado na seção seguinte. Segundo [17], posteriormente surgiram muitas alternativas proprietárias que não utilizam o protocolo OpenFlow, pela justificativa de que algumas necessidades específicas não eram atendidas em tempo por esse padrão. Surgiram, assim, outros padrões abertos como o Lisp e o NetConf, e adaptações de protocolos "consagrados" como OSPF, MPLS e BGP para SDN. Também surgiram padrões proprietários a partir de movimentações de empresas como Cisco, Juniper, Big Switch Networks, Brocade, Arista, Extreme Networks, IBM, Dell e HP, dentre outros, sendo o Cisco OpFlex uma das APIs *southbound* mais populares [11].

[1] A *Open Networking Foundation* é uma organização sem fins lucrativos dedicada ao desenvolvimento, divulgação e adoção de SDN por meio de padrões abertos. Ela foi fundada em 2011 como uma organização de desenvolvimento de padrões, e possui um departamento de divulgação muito ativo que é usado para promover o protocolo OpenFlow e outros esforços relacionados a SDN. A organização hospeda uma conferência anual denominada *Open Networking Summit* como parte desses esforços [34].

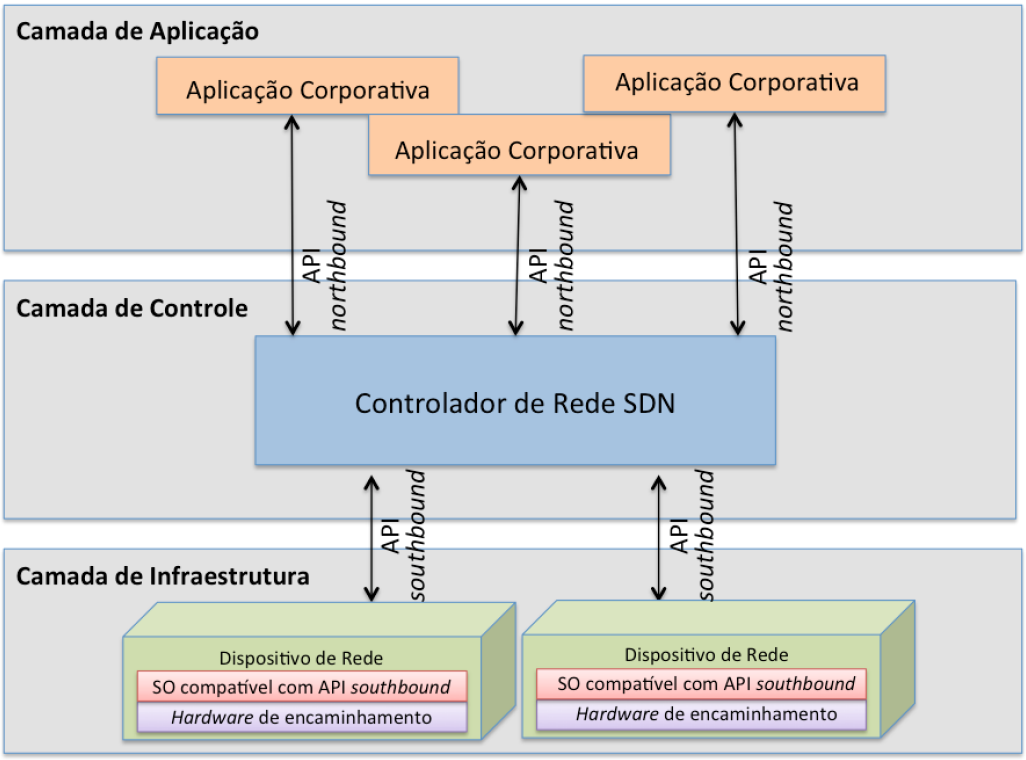



Figura 2.1: Arquitetura de uma rede SDN ilustrando as APIs *northbound* e *southbound*.

### 2.1.2 OpenFlow

O OpenFlow é um padrão aberto de API *southbound* para SDN, que é adicionado como funcionalidade a *switches*, roteadores e pontos de acesso de redes sem fio comerciais [5]. Segundo [33], ele foi originalmente imaginado e implementado como parte da rede de pesquisa na Universidade de Stanford. Seu foco inicial era permitir a criação de protocolos experimentais em redes do campus que poderiam ser utilizadas para a pesquisa e experimentação. Antes disso, as universidades tiveram que criar suas próprias plataformas de pesquisa do princípio. Essa ideia evoluiu para a visão de que o OpenFlow poderia substituir a funcionalidade dos protocolos de camadas 2 e 3 completamente em roteadores comerciais.

O OpenFlow é um padrão de comunicação, e não um produto por si só, ou mesmo uma única característica de um produto. Em sua arquitetura, ilustrada na Figura 2.2, há um plano de controle, composto pelo OpenFlow em si e por um ou mais controladores de rede SDN. Cada controlador atua a partir de um ou mais programas de aplicação que lhe instruem sobre quais fluxos passam por quais elementos e de que maneira. Os controladores, então, se comunicam com interfaces compatíveis com o padrão OpenFlow nos dispositivos de rede para controlá-los. Dessa forma, os *switches* e roteadores da rede passam a ter seus recursos gerenciáveis a partir do controlador. Na Figura 2.2, é ilustrado um exemplo em que o controlador SDN se comunica com as interfaces de alguns roteadores utilizando o padrão OpenFlow para implementar na infraestrutura de rede o protocolo OSPF.

Em suma, uma rede programada com OpenFlow possui pelo menos as seguintes características chave:

- Separação entre plano de controle e plano de dados;
- Utilização de um protocolo padronizado entre o controlador e o agente no elemento da rede;
- Controle de fluxos em dispositivos de rede;
- Programabilidade da rede por meio de uma API centralizada e extensível.

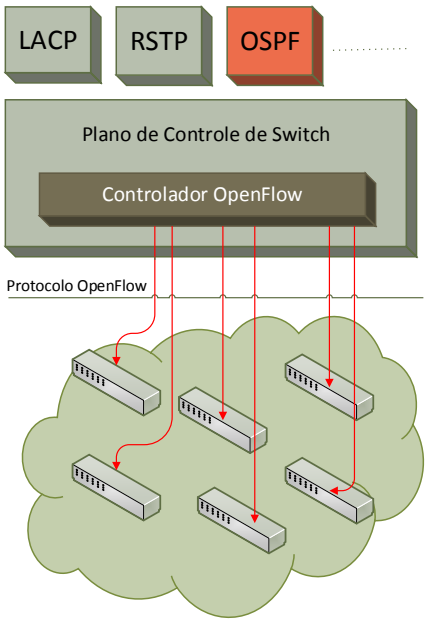



Figura 2.2: Arquitetura do OpenFlow, adaptada de [33].

### 2.1.3 *Frameworks* para SDN

Há diversos *frameworks* para o desenvolvimento de aplicações de controle para SDN, que possibilitam a criação de controladores SDN com diversas funcionalidades e a implementação de uma série de protocolos. O NOX e o POX são dois dos *frameworks* mais comuns. Ambos são disponibilizados de forma aberta em [42], e fornecem módulos de suporte específicos para OpenFlow na versão 1.0, que podem e vêm sendo estendidos para outros padrões e versões. O POX, por exemplo, possui compatibilidade com a extensão da Nicira[2] para funcionamento com o Open vSwitch[3].

Segundo [42], o controlador NOX foi desenvolvido pela Nicira e doado para a comunidade de pesquisa, tendo seu código aberto em 2008. Este movimento fez do NOX um dos primeiros controladores OpenFlow em código aberto. Posteriormente, foi estendido através de uma atividade na Universidade de Stanford. O NOX fornece uma API C++

[2]Empresa americana especializada em SDN e virtualização de redes.

[3]O Open vSwitch é um *switch* compatível com o OpenFlow em *software*. Existem também outros *switches* em *software* e diversos outros *switches* em *hardware* para SDN, como os dos fabricantes Broadcom, Cisco, HP, 3COM e outros.

para OpenFlow 1.0 e um modelo de programação assíncrono baseado em eventos. O núcleo NOX provê métodos auxiliares e APIs para interação com *switches* OpenFlow, incluindo um manipulador de conexão e um mecanismo de controle de eventos. Alguns componentes adicionais que alavancam essa API também são disponibilizados, como um módulo de rastreamento de encaminhamento e topologia (LLDP[4]). Em sua versão clássica, o NOX possui também uma interface em Python implementada como um *wrapper* para a interface programável.

O POX é um *framework* mais recente que foi desenvolvido com base no NOX, com programação em Python. Ele tem uma API SDN de alto nível, incluindo suporte a consultas de topologia e a virtualização, e suporta a mesma interface gráfica e ferramentas de visualização do NOX. Ainda que o POX tenha sido criado a partir do NOX, há algumas diferenças fundamentais entre eles, que tornam o uso do POX possivelmente mais vantajoso. O POX possui componentes reutilizáveis de amostragem para seleção de caminho e descoberta da topologia. Além disso, ele pode ser empacotado para fácil implantação, e é compatível com os sistemas operacionais e plataformas mais populares, podendo ser executado em praticamente qualquer ambiente [42]. Adicionalmente, o POX possui uma interface de configuração em Python que em geral possui um desempenho superior ao das aplicações NOX escritas nessa mesma linguagem. Por essas razões, o POX foi escolhido como controlador para a implementação do protótipo de solução de gerenciamento de desempenho deste trabalho, como será descrito nos capítulos seguintes.

Outros *frameworks* de código aberto que vêm ganhando popularidade são o Ryu, que também é programado em Python e se destaca por ser compatível com uma vasta gama de padrões de SDN, como OpenFlow (1.0, 1.2, 1.3, 1.4 e extensões da Nicira), Netconf e OF-config [45]; e o Floodlight, desenvolvido em Java e compatível com OpenFlow 1.0 e 1.3 [23].

### 2.1.4 Mininet

O Mininet é um emulador de rede SDN que emula uma coleção de hosts, switches, roteadores e links em uma única máquina virtual Linux, executando códigos reais desses dispositivos [32]. Ele utiliza a virtualização para compor um único sistema parecido com uma rede completa, executada no mesmo núcleo, sistema, e código de usuário. O Mininet é importante para a comunidade SDN por ser frequentemente utilizado para simulação, verificação, e experimentação em diversos trabalhos acadêmicos, conforme comentado em [33]. Ele é um projeto de código fonte aberto disponibilizado no GitHub [5].

A Figura 2.3 mostra um exemplo de topologia básica para estudos, sugerida pelo tutorial da página oficial do projeto OpenFlow. Ela contém um *switch* OpenFlow, seu controlador e três hosts conectados a este *switch*. O controlador OpenFlow utiliza uma porta TCP padrão (6633) para se comunicar com o *switch*.

[4]O protocolo *Link Layer Discovery Protocol* (LLDP) é usado por dispositivos de rede para informar suas identidades, capacidades e vizinhos em uma rede local, principalmente em redes Ethernet cabeadas.

[5]O GitHub é um serviço de *web hosting* compartilhado para projetos, que utiliza controle de versionamento.

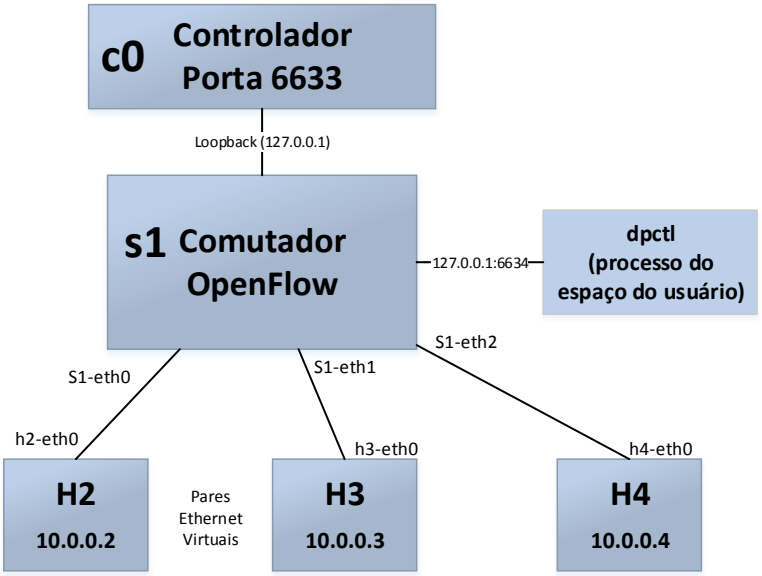



Figura 2.3: Exemplo de topologia criada com Mininet, adaptada de [6].

### Máquina Virtual do Mininet

Está disponível na página oficial do Mininet [32] uma máquina virtual (do Inglês, *Virtual Machine*, VM) para emulações de SDN. Ela é um *appliance* virtual pré-configurado com o Mininet 2.1.0, todos os binários e ferramentas do OpenFlow pré-instalados, e ajustes na configuração do *kernel* para suportar redes Mininet maiores. Essa VM é bastante útil para que usuários compreendam e apliquem configurações em um ambiente virtualizado antes de aplicá-las em ambientes de produção, pois já traz um ambiente pré-configurado para testes.

A VM é executada em um Sistema Operacional Ubuntu 14.04 LTS, com arquitetura 64 bits. Ela inclui uma variedade de utilitários de rede e utilitários específicos do OpenFlow pré-instalados. Os principais utilitários disponibilizados na VM são:

- Controlador OpenFlow: a distribuição de referência inclui a possibilidade de emulação de controladores compatíveis com o padrão OpenFlow. Ao executá-los é possível ver os fluxos de redes que passam por eles. É possível utilizar o NOX já disponibilizado no Mininet e escrever o próprio controlador com base nele, ou "importar" outro *framework* de controle, como o POX;
- Switch OpenFlow: o Mininet disponibiliza *switches* em *software* compatíveis com o OpenFlow no espaço do usuário;
- dpctl: utilitário de linha de comando que envia mensagens rápidas de OpenFlow. É útil para visualizar portas do switch, estados de fluxos e entradas de fluxos inseridas manualmente;
- iperf: utilitário de linha de comando para teste de velocidade de uma única conexão TCP;

- cbench: utilitário para testar a taxa de configuração de fluxo de controladores OpenFlow.

Além disso, foi instalado na VM do Mininet o Wireshark, utilitário gráfico de captura e visualização de pacotes. O Wireshark pode ser configurado para ler mensagens na porta padrão do OpenFlow (6633), de forma a se capturar todos os pacotes que passam pelo controlador de rede SDN.

## 2.2 Middleboxes

Um middlebox é qualquer dispositivo físico ou virtual que intercepta o tráfego e toma ações sobre ele que não são de roteamento ou encaminhamento de pacotes. Middleboxes são, portanto, dispositivos que atuam sobre o tráfego, com o potencial de inspecioná-lo, bloqueá-lo ou modificá-lo. Segundo a RFC 3234 [9], são dispositivos classificados como middleboxes:

- NATs e NAT-PTs;
- *Gateways* SOCKS;
- *Firewalls* IP e de aplicação;
- *Gateways* de aplicação;
- Terminações de túnel IP;
- Classificadores, marcadores e agendadores de pacotes;
- *Relays* de transporte;
- *Proxies*;
- *Caches*;
- Balanceadores de carga;
- *Gatekeepers* / controladores de sessões;
- *Transcoders*;
- Servidores DNS modificados;
- Caixas de distribuição de conteúdo e de aplicações;
- Interceptadores em nível de aplicação (*pseudo-proxies*);
- Dispositivos que promovem *multicast* em nível de aplicação;
- Dispositivos que promovem redirecionamento involuntário de pacotes;
- Dispositivos utilizados para tornar comunicações "anônimas".

Pode-se estender o conceito apresentado para incluir ainda mais dispositivos que se populizaram após a publicação da RFC 3234, como os sistemas de detecção e prevenção de intrusões e os otimizadores de redes de longa distância (do Inglês, *Wide Area Networks*, WANs).

## 2.3 Gerenciamento de Desempenho de Aplicações

O gerenciamento de desempenho de aplicações tem por objetivo permitir aos administradores da TI de uma organização verificarem continuamente o desempenho dos serviços providos a seus usuários finais, identificando gargalos e simplificando o processo de resolução de problemas. Há diversas soluções de mercado com essa proposta, que por vezes agregam a ela funcionalidades de gerenciamento de desempenho de rede e infraestrutura. Segundo [24], as principais funcionalidades que são encontradas no mercado de soluções de APM são:

- Monitoramento de experiência de usuário final, que consiste em verificar como a percepção de usuários finais é afetada em termos de atrasos fim-a-fim, corretude de execução e qualidade [6];
- Descoberta e visibilidade de topologia de aplicação, em que são descobertos os componentes envolvidos na execução de uma aplicação e os possíveis caminhos entre esses componentes para a entrega da aplicação;
- Geração de perfis de transações definidas/agrupadas por usuários, em que é feito o rastreamento de transações com o detalhamento da interação entre os servidores que compõem a aplicação;
- Descoberta de métricas aprofundadas de componentes de aplicação, com o monitoramento e controle granular de recursos e de eventos ocorridos nos servidores que compõem uma aplicação;
- Análise de operações de TI, em que são descobertos/reconhecidos padrões de comportamento das aplicações.

Para prover essas e outras funcionalidades, as soluções de APM podem obter informações do comportamento das aplicações de várias formas. Algumas das mais comuns são:

- A obtenção de dados de disponibilidade e desempenho de infraestrutura via SNMP ou WMI;
- A coleta de informações relativas à comunicação entre os componentes da aplicação a partir de espelhamentos de pacotes (*Port SPAN*), agregadores (TAPs) ou *flows* (como NetFlow e sFlow);
- A coleta de dados de servidores de aplicação e bancos de dados a partir da instalação de agentes específicos;
- A obtenção de dados de experiência de usuário final a partir da instalação de agentes na máquina cliente, ou de alguma instrumentação do serviço acessado (como a inserção de códigos JavaScript em páginas Web);
- A configuração de coletas em dispositivos de rede ou middleboxes.

[6] Por vezes esse monitoramento é realizado a partir de transações sintéticas que simulam usuários finais.

Como comentado anteriormente, o processo de monitoramento pode ser trabalhoso devido à grande quantidade de pontos de coleta necessários para se obter informações válidas de aplicações complexas, à diversidade de equipamentos presentes na infraestrutura (especialmente middleboxes), e à mutabilidade/dinamismo da infraestrutura como um todo, em termos de componentes e de configurações.

## 2.4 Revisão do Estado da Arte

O tema de Redes Definidas por Software tem recebido crescente atenção da indústria e da comunidade científica nos últimos anos. Devido ao conceito de SDN se aplicar diretamente apenas às camadas 2 e 3 do modelo OSI, a maioria das propostas e inovações em SDN se dão nesse contexto. Há diversas propostas de arquitetura de redes para SDN, de padrões similares ou complementares ao OpenFlow, e de integração de redes SDN com redes tradicionais. Já para as camadas de 4 a 7 do modelo OSI, segmento em que se concentram a maior parte das funcionalidades dos middleboxes e a maioria das métricas que compõem gerenciamento de desempenho de aplicações, há um desenvolvimento mais tímido.

Existem algumas publicações propondo middleboxes próprios para SDN, como [55], [47] e [53], que propõem modelos de balanceadores de carga próprios para SDN. Essas publicações trazem mecanismos de balanceamento para OpenFlow que são implementados a partir da codificação de uma funcionalidade do controlador SDN. Em [55], é desenvolvido um balanceador com um controlador NOX que trabalha com *wildcards* nas regras de balanceamento de carga, de forma que a distribuição de tráfego seja mais flexível e escalável. Em [47], é proposta uma solução de balanceamento com um controlador Floodlight, capaz de distribuir o tráfego em três mecanismos: aleatório, *Round-robin* e baseado na carga do servidor. Em [53], esses mesmos mecanismos são implementados e avaliados em um controlador NOX. Além disso, há exemplos de soluções comerciais de middleboxes que são compatíveis com SDN, como o Virtual Traffic Manager [8], da fabricante Brocade, o BIG-IP [18], fabricado pela F5, o NetScaler [14], desenvolvido Citrix, e o FastView [41], da Radware.

Há também algumas propostas para melhorar a configuração dos middleboxes em SDN, como [25], [50], [15] e [46]. Em [25], é proposto um *framework* para redes com middleboxes em SDN, contendo um mecanismo para se exercer controle unificado sobre os fatores que influenciam as operações de middleboxes. Esse mecanismo possibilitaria o gerenciamento de ambientes complexos que contenham middleboxes. Em [50], é proposto um protocolo para configuração dinâmica para SDN de dois exemplos comuns de middleboxes, os NATs e os *firewalls*, denominado *simco*. O *simco* pretende fazer com que aplicações se comuniquem com middleboxes no caminho de seus datagramas para requisitar configurações dinâmicas relativas a seus fluxos de dados.

Já em [15], [46] e [3], são propostos novos modelos de desenvolvimento de middleboxes que poderiam ser aplicados em SDN. Em [15], é proposto um gerenciador de tráfego e de recursos de rede centralizado, a partir do qual funcionalidades de middleboxes como o NAT poderiam ser implementados. Em [46], é apresentada uma arquitetura para desenvolvimento de middleboxes consolidados. Em [3], é proposta uma arquitetura de middleboxes baseada em servidores e sistemas operacionais *commodity*, que possui módulos em C++ para realizar todas as funcionalidades de análise e modificação de tráfego.

Em relação ao gerenciamento de desempenho envolvendo middleboxes para SDN, foram encontrados poucos trabalhos publicados. O artigo [40] discute uma proposta para melhorar o gerenciamento de middleboxes em SDN sem trazer limitações de localização ou de implementação de funcionalidades, a partir dos padrões de SDN já disponíveis. No entanto, a proposta não é elaborada ou implementada em [40], que é um artigo introdutório indicando apenas como os autores pretendem abordar este tema. Em [25], conforme explicou-se anteriormente, também há algum trabalho no sentido de melhorar o gerenciamento de middleboxes, mas o gerenciamento de desempenho de aplicações em redes SDN com middleboxes não é abordado em nenhuma publicação encontrada.

Avaliando-se as soluções de mercado para gerenciamento de desempenho de aplicações compatíveis SDN, encontrou-se apenas a solução Riverbed SteelCentral [43]. No entanto, o SteelCentral é uma ferramenta paga, e os detalhes de sua implementação não são abertos ao público geral. Assim, com essa revisão de estado da arte, pôde-se observar que há alguns esforços direcionados para o tema proposto, mas que há pouquíssimas publicações envolvendo diretamente esse tema, e apenas uma solução de mercado compatível com o que esse trabalho se propõe a fazer. Conclui-se, portanto, que existe uma grande oportunidade de desenvolvimento do tema proposto.

# Capítulo 3

# Arquitetura de Gerenciamento de Desempenho

É apresentada neste capítulo uma arquitetura de gerenciamento de desempenho de aplicações adaptada às Redes Definidas por Software, com foco em infraestruturas com middleboxes. Seu intuito é servir como um modelo de funcionamento para soluções de APM, de forma que elas sejam capazes de interpretar informações de middleboxes de forma universal, e de funcionar em ambientes SDN. A arquitetura proposta almeja ir além da compatibilidade com SDN, buscando beneficiar-se das características de SDN para simplificar o processo de monitoramento/gerenciamento. Além disso, ela propõe um mecanismo de comunicação padronizado por tipo de middlebox, para contornar as limitações que a grande diversidade no âmbito dos middleboxes costumeiramente traz às soluções de APM. A proposta sugere, ainda, uma forma de consolidar esses dados e prover informações de monitoramento de aplicações aos administradores.

A metodologia do trabalho realizado e a modelagem conceitual da arquitetura proposta são descritas a seguir. Para a avaliação da viabilidade da arquitetura, seus conceitos são implementados em um protótipo de gerenciamento de desempenho. Esse protótipo se propõe a servir como uma base para o desenvolvimento de soluções de APM mais completas para SDN, tratando redes com ou sem a presença de middleboxes, mas mais voltado para o último caso. A implementação e a validação do protótipo serão detalhadas nos capítulos seguintes.

## 3.1 Metodologia

Para a elaboração da arquitetura de gerenciamento de desempenho e o desenvolvimento e validação do protótipo, foram realizadas as seguintes etapas:

1. Determinação do modelo de funcionamento da arquitetura: elaboração de um modelo de entradas e saídas para a arquitetura proposta, incluindo uma descrição conceitual de como seria realizada a obtenção e disponibilização de informações no modelo;
2. Escolha de mecanismo de obtenção de dados do(s) controlador(es): determinação de um modelo de comunicação entre o(s) controlador(es) SDN e a solução de gerenciamento de desempenho;

3. Modelagem das relações de middleboxes e métricas: revisão de middleboxes de uso mais comum, e determinação de quais métricas devem ser obtidas de cada middlebox para que se disponibilizem as saídas desejadas;

4. Determinação de mecanismo de comunicação da solução proposto com os middleboxes: verificação de qual mecanismo de comunicação entre os middleboxes e a solução de gerenciamento seria mais adequado;

5. Definição de estratégia de armazenamento e consolidação dos dados: proposta de um mecanismo que consolide os dados obtidos dos middleboxes e do(s) controlador(es) em informações úteis para a administração do desempenho das aplicações, e armazene essas informações para consultas posteriores;

6. Implementação do protótipo: desenvolvimento de protótipo de solução de APM que utilize a arquitetura proposta;

7. Validação: planejamento e execução de uma sequência de avaliações para verificar a viabilidade da arquitetura proposta a partir do protótipo implementado.

Os trabalhos e resultados obtidos nessas etapas da metodologia são descritos nas seções subsequentes.

## 3.2 Modelo de Funcionamento da Arquitetura

A arquitetura aqui descrita foi pensada para otimizar o monitoramento de aplicações a partir de duas características de SDN, a centralização do controle e a programabilidade. Seu intuito não é substituir todas as formas de monitoramento/gerenciamento já praticadas atualmente, mas sim complementá-las e servir como um modelo para a adaptação/criação de soluções de APM para SDN de forma otimizada. A Figura 3.1 ilustra de forma simplificada o seu modelo de funcionamento.

Com a separação do plano de controle em uma camada com dispositivos próprios e especializados, o paradigma de SDN permite que um grande conjunto de informações da rede se concentre em pontos centrais, os controladores de rede. Isso significa que as características dos fluxos da rede podem ser obtidas a partir da interação com esses pontos centrais, ao invés de serem obtidas de forma descentralizada (por exportação de fluxos de rede em protocolos como o NetFlow, espelhamentos de rede por SPAN/RSPAN, dentre outros), o que naturalmente seria uma tarefa mais exaustiva. O modelo de arquitetura proposto compreende a comunicação entre a solução de APM e o(s) controlador(es) da rede SDN para a obtenção das informações de fluxos que trafegam pela rede.

A partir das informações de fluxos de um controlador, vários dados de conexões e pacotes podem ser determinados. São exemplos desses dados a quantidade de *bytes* trocados em uma comunicação, os endereços IP e MAC de origem e destino, e os protocolos de comunicação utilizados, dentre várias outras informações que podem ser verificadas com a análise dos cabeçalhos dos pacotes. Com uma análise um pouco mais aprofundada dos pacotes recebidos, é possível determinar outros dados de conexão que podem ser interessantes para o gerenciamento de desempenho de aplicações, como a presença de *resets* e retransmissões (que naturalmente impactam o desempenho), o estado da conexão (por exemplo, uma resposta 200 OK em uma conexão HTTP, indicando o sucesso na

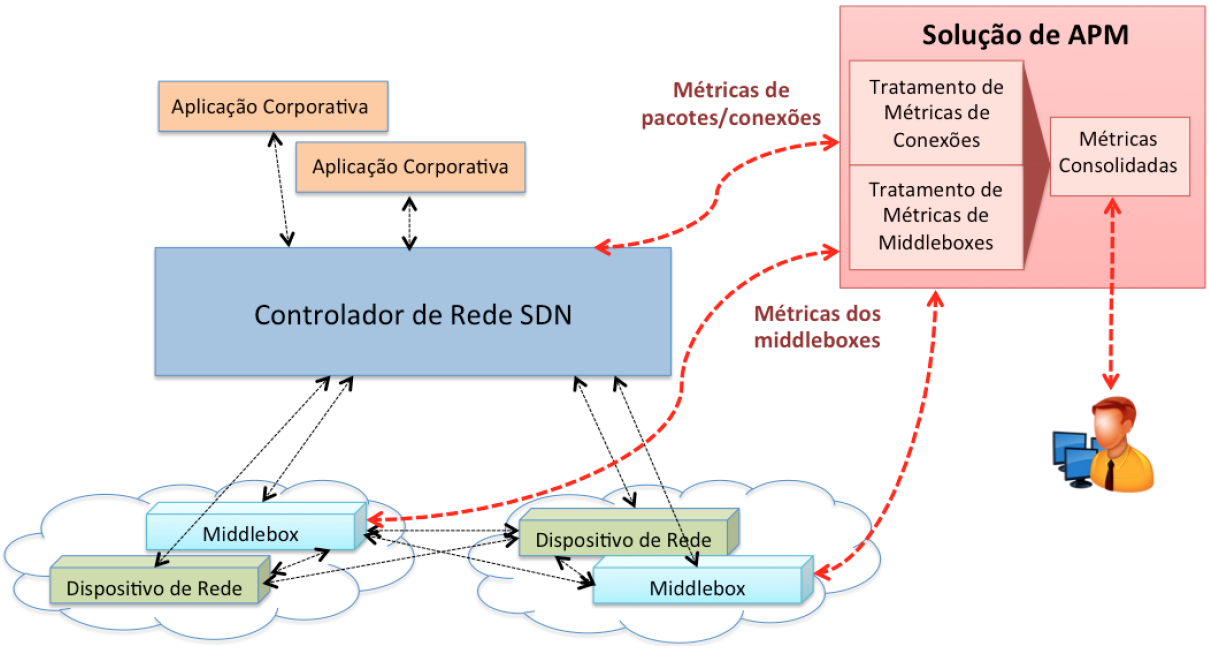



Figura 3.1: Modelagem conceitual da arquitetura da solução de APM proposta.

requisição), e a duração das conexões, que influencia diretamente o tempo de resposta da aplicação[1]. O modelo de comunicação com o controlador e os dados obtidos a partir dessa comunicação são detalhados na Seção 3.3.

Essa interação com o controlador deverá trazer uma série de informações de rede interessantes ao gerenciamento de desempenho das aplicações, partindo de uma obtenção de dados bastante prática em relação a um "*sniffing*" de pacotes distribuído. No entanto, todas as limitações provocadas pela presença de middleboxes ao gerenciamento continuariam presentes caso a solução proposta se limitasse a essa interação. À obtenção e tratamento dos dados de fluxos de redes do controlador, é adicionada no modelo a comunicação da solução com os middleboxes presentes na infraestrutura. Para a modelagem dessa comunicação, é necessário determinar de que forma cada tipo de middlebox pode interferir sobre o tráfego da aplicação, e transformar essa informação em um conjunto de estados e métricas que possa ser tratado pela solução de APM. Essa determinação é detalhada na Seção 3.4.

Uma vez determinado um conjunto de métricas e estados que informem de forma suficiente a atuação dos middleboxes sobre a aplicação, é preciso determinar a forma de comunicação dos middleboxes com a solução de APM. É essencial que essa comunicação seja pouco intrusiva (exija pouca ou nenhuma alteração do ambiente monitorado) e que imponha um *overhead* mínimo ao ambiente. A opção de comunicação escolhida é descrita na Seção 3.5. Por fim, é necessário consolidar os dados obtidos do(s) controlador(es) e dos middleboxes. Essa consolidação, detalhada na Seção 3.6, deve ser capaz de unificar métricas de uma mesma conexão ou transação de aplicação, para que a solução de

[1]O tempo de resposta de uma transação em uma aplicação cliente-servidor é uma composição dos atrasos de rede, servidor e cliente. Para aplicações cliente-servidor que não têm um grande tempo de processamento no cliente, o tempo de resposta pode ser aproximado para a duração da comunicação cliente-servidor que compreende a transação analisada.

APM de fato atenda o propósito de contornar as limitações impostas por middleboxes ao gerenciamento.

## 3.3 Comunicação com Controladores SDN

Há múltiplas abordagens possíveis para a interação com os controladores de redes SDN, dependendo de quais dados dos fluxos de redes a solução procura obter. Uma abordagem é trabalhar de forma não-intrusiva a partir da inspeção dos *logs* do controlador, cujas informações variam a depender do controlador utilizado. No entanto, em geral poucos detalhes sobre os fluxos são fornecidos em *logs*, ao menos nos controladores mais comuns. Assim, possivelmente não seriam obtidos dados suficientes para as análises pretendidas. Outra abordagem possível seria modificar o código do controlador para que sejam enviados os dados requisitados da forma mais "leve" possível. Essa abordagem, porém, é consideravelmente intrusiva. Dificilmente em uma rede SDN de produção seria aceitável modificar o controlador em si para o gerenciamento de desempenho, visto que o controlador é o "coração" da rede, e todo o funcionamento da rede é diretamente impactado por sua responsividade.

Outra abordagem possível seria trabalhar com uma captura ("*sniffing*") de pacotes centralizada em um ou mais controladores da rede SDN, dependendo de quais dos controladores atuam sobre segmentos de rede em que trafegam as aplicações a serem monitoradas. Essa captura pode ser feita sobre o tráfego dos controladores virtualmente das mesmas maneiras que nos *switches* e roteadores tradicionais. Existem vários programas livres que realizam "*sniffing*" de redes e que poderiam ser utilizados neste caso, como o tcpdump, o dumpcap e o Wireshark. Com as capturas completas, a flexibilidade na obtenção de informações de pacotes e conexões relacionados a uma aplicação é muito maior, visto que todos os cabeçalhos dos pacotes estarão disponíveis, e eventualmente até mesmo seus *payloads*, caso se considere necessário para alguma análise específica. Logicamente, para reduzir o *overhead* dessas capturas de tráfego, podem ser definidos filtros para as capturas, funcionalidade suportada por todos os programas de captura citados.

Já existem diversas soluções de gerenciamento de desempenho de redes e aplicações que trabalham com capturas de tráfego. Assim, seria simples adaptá-las a esse modelo de comunicação com os controladores de rede SDN. No caso do desenvolvimento de uma solução de APM nova, ela poderia trabalhar em conjunto com *softwares* já estabelecidos nesse segmento para realizar sua análise, como o *software* aberto de análise de pacotes Wireshark ou sua adaptação em linha de comando TShark [52], e as soluções proprietárias Riverbed Packet Analyzer [2] e Fluke Visual TruView [51]. Devido à opção de "*sniffing*" de pacotes dos controladores ser pouco intrusiva e prover um conjunto de informações mais completo sobre os fluxos de redes, ela foi escolhida para o modelo de arquitetura proposto. O *overhead* imposto por essa opção é regulável pelos filtros utilizados na captura.

## 3.4 Middleboxes e Métricas

Para atender os propósitos da arquitetura de gerenciamento sugerida, foi determinado um conjunto de estados/métricas a serem obtidos de middleboxes que informam a sua interferência no tráfego de uma dada aplicação. Buscou-se obter um conjunto mínimo

suficiente para o gerenciamento, de forma a minimizar o *overhead* de transmissão dessas informações dos middleboxes para a solução de APM. A partir da revisão bibliográfica realizada e do conhecimento prático adquirido, selecionou-se, dentre os tipos de middleboxes existentes, o seguinte conjunto: *firewall*, sistema de tradução de endereços (NAT) ou de portas de rede (do Inglês, *Port Address Translation*, PAT), balanceador de carga, sistema de detecção/prevenção de intrusão (do Inglês, *Intrusion Detection/Prevention System*, IDS/IPS), e sistema de eliminação de redundância (*cache/proxy*). O critério para essa escolha foi a verificação do quão comum é a presença de cada tipo de middlebox nas redes corporativas, e de quais tipos de middleboxes de fato têm influência significativa no funcionamento e no desempenho de uma aplicação.

De forma semelhante a [25], foi selecionado um conjunto de características que definem um estado de cada um desses tipos de middleboxes, tendo em vista a sua influência em parâmetros de desempenho de aplicações. A Tabela 3.1 resume os middleboxes e os respectivos parâmetros sugeridos nesta modelagem. Por exemplo, a disponibilidade de um middlebox pode influenciar diretamente a disponibilidade de uma aplicação. Portanto, a métrica de estado do middlebox (ativo/inativo) é recolhida de todos os middleboxes. Similarmente, o tratamento dos pacotes de uma conexão da aplicação pelo middlebox pode influenciar o comportamento de uma aplicação em termos de eficiência ou de corretude de funcionamento. Assim, também são recolhidos de cada middlebox os registros que distinguem uma conexão (endereços IP de origem e destino, portas de origem e destino e horário de início), bem como as informações que determinam a atuação do middlebox sobre a conexão.

As variáveis adicionais a serem enviadas dependem do tipo de atuação do middlebox. No caso de um firewall, sua atuação pode ser de bloquear ou liberar uma conexão, portanto seria enviada uma variável indicando o estado de conexão correspondente (bloqueada/liberada). Para um NAT, sua atuação é de mapeamento, sendo enviada uma variável em que conste o registro de mapeamento (endereço de rede original/endereço de rede mapeado). Um PAT enviaria as mesmas variáveis, mas com o registro de mapeamento de portas (porta original/porta mapeada). Um balanceador de carga deveria indicar qual servidor selecionou para atender à requisição, qual o algoritmo de balanceamento utilizado em sua seleção (para que se possa determinar se a escolha um algoritmo específico está afetando o desempenho), e qual o estado do *pool* de servidores a ele conectado (caso o *pool* se torne indisponível, a disponibilidade da aplicação também é afetada). Já um IPS deveria indicar se a conexão foi bloqueada ou gerou algum tipo de alerta, parâmetro que foi denominado nível de alerta/bloqueio. O parâmetro é basicamente o mesmo para um IDS, que também pode gerar alertas sobre o tráfego, ainda que não o bloqueie. Para um sistema de eliminação de redundância, como um *cache* ou um *proxy*, deve ser informado o registro de substituição dos dados. Por exemplo, se uma requisição foi redirecionada a um servidor específico ou se foi atendida localmente pelo sistema, esse registro deve ser informado à solução de APM.

Recebendo o conjunto de variáveis exposto na Tabela 3.1, a solução de APM deverá ser capaz de compreender a atuação de cada middlebox sobre cada conexão do tráfego de uma aplicação. Assim, essas informações serão cruciais para que a solução de APM consiga prover informações de desempenho de aplicações válidas mesmo com a presença de múltiplos middleboxes, contornando as dificuldades impostas por middleboxes já comentadas. Além disso, o conjunto de métricas proposto é bastante reduzido, para que o

envio contínuo dessas métricas seja leve.

| **Middlebox** | **Definição de Estado** |
|---|---|
| *Firewall* | Registros de conexão (IPs, portas, horário), estado do *firewall*, estado da conexão (bloqueada/liberada). |
| NAT | Registros de conexão (IPs, portas, horário), estado do NAT, registro de mapeamento de endereços. |
| PAT | Registros de conexão (IPs, portas, horário), estado do PAT, registro de mapeamento de portas. |
| Balanceador de Carga | Registros de conexão (IPs, portas, horário), estado do balanceador, estado do *pool* de servidores, servidor de destino da conexão, algoritmo de balanceamento. |
| IDS/IPS | Registros de conexão (IPs, portas, horário), estado do IDS/IPS, nível de alerta/bloqueio. |
| Eliminação de Redundância (*Cache/Proxy*) | Registros de conexões (IPs, portas, horário), estado do *cache/proxy*, registro de substituição dos dados. |

Tabela 3.1: Levantamento de middleboxes e métricas mais relevantes para definição dos estados.

## 3.5 Comunicação com Middleboxes

A interação da solução de APM com os middleboxes deve ser realizada de forma ágil, para que as informações recolhidas deles sejam disponibilizadas em tempo próximo ao real, e impondo o mínimo de *overhead* aos middleboxes e à rede possível. Considerando-se a grande proporção de middleboxes nas redes corporativas [35], é essencial que a comunicação dos middleboxes com a solução seja leve. Para atender a esse propósito, o modelo sugere a utilização do formato de intercâmbio de dados *JavaScript Object Notation* (JSON). Esse formato é leve, de fácil interpretação e universal (é um formato de texto independente de linguagens de programação [30]). Uma variável JSON é enxergada como uma estrutura de dados composta por um conjunto de pares nome-valor. No contexto da modelagem proposta, o conjunto de parâmetros de cada middlebox (conforme descrito na Tabela 3.1) seria enviado para a solução de APM dentro de uma estrutura JSON, com seus respectivos nomes e valores, a cada conexão que trafegue pelo middlebox.

Além do baixo *overhead* de comunicação, a solução de APM deve impor pouca ou nenhuma mudança às middleboxes, para que seja de fato adaptável a ambientes reais. O conceito de programabilidade das redes SDN é compatível com esse requisito. Dado que o paradigma de SDN propõe que todos os dispositivos de uma rede disponham de APIs abertas, é razoável assumir que os middleboxes compatíveis com SDN também possuam um certo nível de programabilidade, disponibilizando interfaces programáveis para a interação com aplicações, o controlador SDN e/ou outros dispositivos. De fato, já existe uma ampla gama de APIs de SDN disponibilizados por uma série de fabricantes,

tanto para dispositivos de redes quanto para middleboxes. A Tabela 3.2 ilustra algumas das principais APIs de SDN presentes no mercado.

Alguns dos middleboxes compatíveis com SDN possivelmente já possuem APIs que poderiam ser aproveitadas pela arquitetura proposta, ainda que não necessariamente utilizem o formato JSON. Por exemplo, as APIs iControl, iCall e iRest listadas na Tabela 3.2, aplicadas aos serviços de aplicação e *gateway* da F5, já provêem a programabilidade a partir de controladores e de aplicações externas de middleboxes disponibilizadas por esse fabricante, como o balanceador de carga BIG-IP [26].

Nos casos em que não houver API disponível no middlebox a ser monitorado, é necessária alguma adaptação para a coleta dos parâmetros elencados. Como os parâmetros escolhidos são bastante simples, comuns e agnósticos a fabricantes, frequentemente poderão ser extraídos dos próprios *logs* do middlebox, à medida em que forem armazenados. A maioria dos sistemas gera *logs* referentes à sua atuação à medida em que são executados, então em tese não haveria atrasos significativos em se obter os parâmetros a partir dos *logs*, ao invés de utilizar uma API diretamente implementada no código do middlebox. Além disso, a opção de monitoramento dos *logs* seria menos intrusiva, pois não oneraria o middlebox em termos de processamento.

Em último caso, se não for possível utilizar APIs já desenvolvidas ou monitorar os parâmetros a partir dos *logs*, sugere-se uma modificação mínima no código do middlebox para o envio dessas informações à solução de APM. No Capítulo 4, detalham-se implementações reais para as duas últimas situações (monitoramento de *logs* e modificação do código).

| **Fabricante** | **API SDN** |
|---|---|
| A10 | aXAPI |
| Arista | EOS API |
| Brocade | ADX OpenScript API |
| Cisco | onePK Nexus 1000V XML API |
| Dell/Force10 | Open Automation Framework |
| Enterasys | OneFabric Connect API |
| Extreme | InSite SDK |
| F5 | iControl, iCall, iRest, REST |
| Juniper | Junos SDK, XML API e Junos Scripting Junos Space |
| OpenStack | Quantum API |
| VMware | VMware vSphere Management SDK |

Tabela 3.2: Levantamento de APIs de SDN de mercado mais relevantes e seus respectivos fabricantes [10] [26].

## 3.6 Armazenamento e Consolidação dos Dados

Uma vez coletadas as informações de rede do(s) controlador(es) e as informações dos middleboxes, é necessário "cruzar" os dados de todas as conexões de forma consistente e pouco onerosa à solução de APM, uma vez que ela deve ser capaz de processar uma grande quantidade de conexões para ser aplicável a uma rede real. Como todos os middleboxes enviam a cada conexão os registros dela (além dos parâmetros adicionais de monitoramento), essa informação pode ser utilizada para que se encontrem os dados do controlador SDN para a mesma conexão. Ou seja, cada registro de conexão (*IP_ origem, Porta_ origem, IP_ destino, Porta_ destino*) unido ao parâmetro *Horário* de início daquela conexão deve estar presente tanto no conjunto de dados de um middlebox quanto na captura de um controlador SDN, e distinguir cada conexão de forma exata. Para que se considerem eventuais diferenças nos horários do middlebox e do controlador, provocadas por possíveis atrasos no envio dos dados pelo middlebox e/ou na captura do controlador, deve-se flexibilizar um pouco o parâmetro de *Horário*. Assim, sugere-se que o requisito de consolidação dos dados de cada conexão seja (*IP_ origem, Porta_ origem, IP_ destino, Porta_ destino,* $Horário \in Intervalo$), onde o *Intervalo* equivale a um dos horários obtidos "relaxado" por uma margem de erro $\epsilon$ ($Horário - \epsilon < t < Horário + \epsilon$).

Para que a consolidação das informações e a sua busca sejam realizadas de forma ágil, o modelo prevê o uso de um banco de dados para o seu armazenamento, considerando um Sistema de Gerência de Banco de Dados (SGBD) relacional[2]. Esse banco de dados deve possuir uma tabela em que as linhas têm a função de armazenar as informações de uma conexão, sendo cada coluna um parâmetro coletado da captura ou de um middlebox. A Figura 3.2 explicita as entidades e relacionamentos do banco de dados proposto, utilizando uma modelagem na notação Engenharia de Informações ou James Martin [28]. Cada linha da tabela compreenderia as seguintes colunas:

- As colunas *IP_ origem, Porta_ origem, IP_ destino, Porta_ destino, Horário*, que compõem o parâmetro de consolidação, como chave primária composta[3];
- As colunas contendo os parâmetros de conexão coletados exclusivamente do controlador. Exemplos de colunas seriam: número de *bytes* trocados na conexão, duração da conexão, protocolo de camada de transporte utilizado;
- As colunas contendo as informações coletadas exclusivamente dos middleboxes. Por exemplo, em uma rede contendo um *firewall* e um NAT, cada parâmetro comporia uma coluna adicional, seguindo a proposta da Tabela 3.1. Assim, seriam adicionadas as seguintes colunas: estado do *firewall*, estado da conexão no *firewall* (bloqueada/-liberada), estado do NAT, registro de mapeamento de endereços do NAT.

Com esta modelagem de banco de dados em consideração, a arquitetura proposta deverá trabalhar da seguinte forma: a cada nova conexão em qualquer dos controladores de rede SDN, é inserida uma nova linha na tabela do banco de dados que lista as conexões.

[2]O modelo de banco de dados relacional é um dos mais praticados atualmente. Nesse modelo, um banco de dados é composto de tabelas, que são conjuntos não-ordenados de linhas. Essas linhas possuem campos, sendo que o conjunto de campos que possuem o mesmo nome compõem uma coluna.

[3]A chave primária composta é um conjunto de colunas que distingue de forma mínima e inequívoca uma linha do banco de dados (no caso, uma conexão).

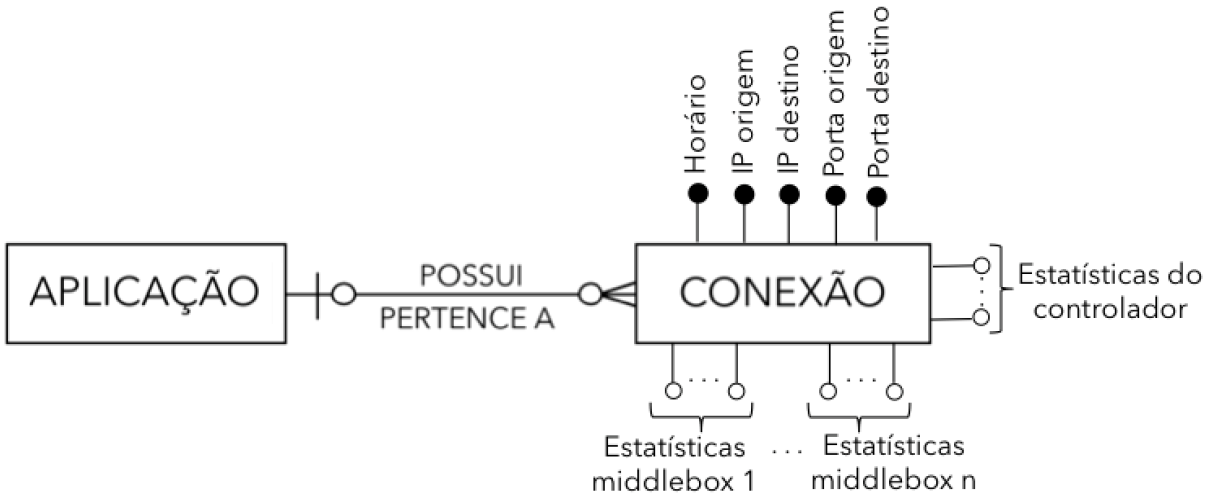


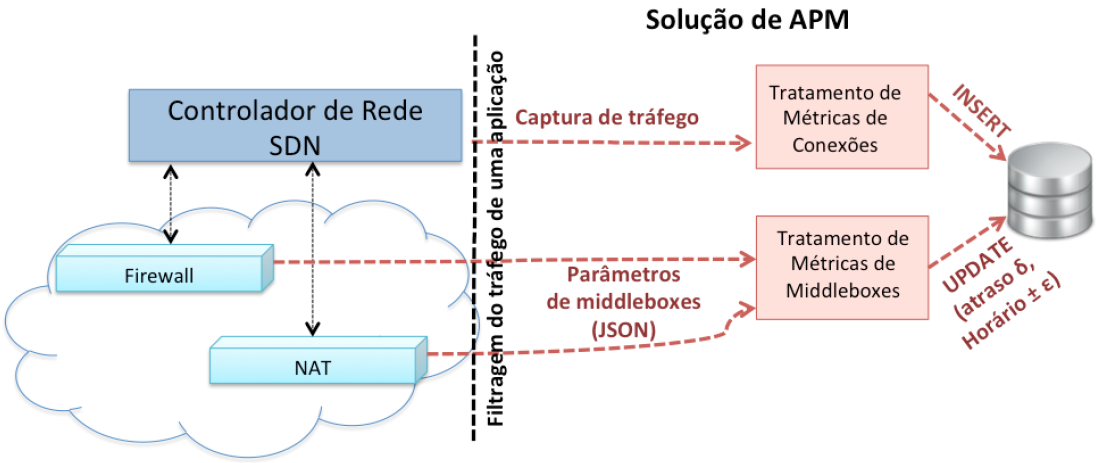

Figura 3.2: Modelagem do banco de dados, utilizando a notação Engenharia de Informações [28].

A partir de filtros de captura configurados pelo administrador da solução, essas conexões são relacionadas à aplicação correspondente (ou a nenhuma aplicação, caso não haja aplicação cadastrada que dê "match" em seu filtro). A linha inserida conterá a chave primária (colunas *IP_ origem, Porta_ origem, IP_ destino, Porta_ destino* e *Horário*) e os demais parâmetros de conexão que forem considerados úteis (por exemplo, *Bytes trocados* e *Duração*), mantendo-se os parâmetros de middleboxes vazios (NULL).

À medida que novas conexões trafeguem por um middlebox qualquer da rede, é feita uma atualização no banco, com um certo atraso $\delta$ para garantir que essas conexões já tenham sido inseridas no banco a partir da coleta dos controladores. Busca-se a chave primária da conexão no banco (considerando-se a "margem de erro" $\epsilon$ para a variável *Horário*), e na linha correspondente àquela conexão, são adicionados os parâmetros de middleboxes. Por exemplo, no caso do *firewall*, seria adicionado o parâmetro de estado do *firewall* e de estado da conexão (bloqueada/liberada). Esse procedimento é ilustrado na Figura 3.3. Todas as conexões são armazenadas no banco de dados por um certo período de retenção definido pelo administrador, para permitir consultas históricas, possíveis comparações entre períodos de tempo, e eventualmente a visualização de tendências de comportamento.

## 3.7 Implementação e Validação do Protótipo

O protótipo de solução de APM tem por objetivo demonstrar o funcionamento da arquitetura proposta e a viabilidade de sua implementação em ambientes reais. Para o seu desenvolvimento, foi utilizado o ambiente de emulação Mininet, por sua simplicidade de obtenção e configuração e por ser um ambiente de testes utilizado em vários trabalhos acadêmicos. O padrão OpenFlow de comunicação entre os controladores e dispositivos de rede foi escolhido, por ser o primeiro e o mais comum dos padrões abertos para SDN, e pelo fato de o Mininet já ser adaptado a esse padrão, incluindo uma série de comandos que simplificam sua implementação.




| IP orig. | IP Dest. | Porta orig. | Porta dest. | Horário início | Estado Firewall | Estado Con. Firewall | Estado NAT | Mapeamento NAT | Bytes na conexão | Duração da conexão (ms) |
|---|---|---|---|---|---|---|---|---|---|---|
| 10.0.0.15 | 172.16.10.12 | 53290 | 80 | 20/06/2015 10:23:03.40 | Ativo | Liberada | Ativo | 10.0.0.1 → 172.16.10.1 | 3416 | 112 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Figura 3.3: Atuação da solução de APM em uma rede com um NAT e um *firewall*. Uma tabela de conexões de uma aplicação é apresentada como exemplo.

Nesse ambiente, o protótipo de gerenciamento de desempenho desenvolvido deve ser capaz de monitorar duas métricas de aplicação principais, o tempo de resposta e a disponibilidade, e também algumas métricas de conexões individuais, a saber: quantidade de *bytes* trafegados, endereços IP de origem e destino, portas de origem e destino, horários de início e fim, duração, e as demais métricas de middleboxes listadas na Tabela 3.1. A implementação do protótipo é discutida em maiores detalhes no Capítulo 4.

A validação do protótipo de solução de APM, por sua vez, tem a intenção de demonstrar se os parâmetros computados pelo protótipo são corretos, e a estrutura desenvolvida é aplicável a um ambiente real. Ela consiste em etapas intermediárias de avaliação, em que o funcionamento de módulos individuais é testado, e em avaliações finais, em que a solução como um todo tem seus resultados avaliados para diferentes topologias de redes SDN, contendo middleboxes diversos. As etapas de validação e seus resultados são discutidos no Capítulo 5.

# Capítulo 4

# Protótipo de Gerenciamento de Desempenho

O protótipo de gerenciamento de desempenho de aplicações foi implementado em uma máquina virtual com o ambiente de emulação Mininet 2.1.0, disponibilizada em [32], que possui Sistema Operacional Ubuntu 14.04 LTS. Nessa máquina, foram criadas topologias em que um *host* específico desempenha o papel da solução de APM, seguindo a arquitetura descrita no Capítulo 3. Todas as topologias utilizavam um controlador POX, disponibilizado em [42], e um ou mais *switches* do tipo Open vSwitch, que se comunicam pelo padrão OpenFlow com o controlador. A solução de APM foi desenvolvida majoritariamente na linguagem de programação Python, por sua simplicidade de desenvolvimento e execução e por sua portabilidade, utilizando também um banco de dados PostgreSQL para armazenar as informações obtidas e alguns *scripts shell*. O *host* que executa a solução de APM é responsável por coletar os dados dos controladores e dos middleboxes, tratá-los e a partir deles disponibilizar duas das principais métricas de aplicações específicas, o **tempo de resposta** e a **disponibilidade**. Para que o *host* possa identificar e filtrar os dados de aplicações específicas, o administrador deve informar os endereços IP dos servidores da aplicação e as portas de destino em que se comunicam.

O *host* disponibiliza ao administrador também informações de rede relativas a cada conexão de uma determinada aplicação que trafegue sobre a rede. Essas informações, que compreendem a **quantidade de *bytes* trafegados, endereços IP de origem e destino, portas de origem e destino, horários de início e fim e duração**, são obtidas a partir de análise das capturas de pacotes do controlador. Além dessas informações de rede, cada conexão é parametrizada também de acordo com os middleboxes pelos quais trafega, procurando seguir o indicado na Tabela 3.1. A única mudança em relação aos parâmetros da Tabela 3.1 é que os estados dos middleboxes em si não ocupam colunas específicas na base de dados que lista as conexões e seus parâmetros. Ao invés de se criar uma coluna adicional na tabela das conexões para indicar se um middlebox está ativo ou inativo, esse estado é deduzido a partir das demais métricas que ele envia. Caso todas as colunas de uma conexão que deveriam conter informações de determinado middlebox estejam vazias (NULL) e essa conexão de fato tenha chegado ao middlebox, o protótipo deduz que o middlebox está inativo durante o intervalo de duração da conexão. Caso uma ou mais colunas tenham sido "preenchidas", o protótipo entende que o middlebox estava ativo naquele intervalo, pois foi capaz de se comunicar satisfatoriamente com o protótipo.

Para o tratamento de informações de middleboxes pelo protótipo, foram considerados apenas middleboxes de código aberto, devido à dificuldade em se obter licenças de demonstração para fins de pesquisa, e também à falta de flexibilidade em se trabalhar com códigos proprietários. São eles:

- Um ***firewall*** que funciona como uma extensão em Python para o controlador POX. Esse sistema foi desenvolvido neste trabalho como um *firewall* de camada 4 baseado no *firewall* de camada 2 disponibilizado em [20][1]. Ele envia para o protótipo os registros de cada conexão e o estado da conexão perante o *firewall* (bloqueada/liberada);
- Um **balanceador de carga** que também funciona como uma extensão em Python para o controlador POX, que é uma versão adaptada do balanceador disponibilizado em [42]. Ele envia para o *host* do protótipo os registros de cada conexão, o estado do *pool* de servidores entre os quais ocorre o balanceamento, o endereço IP do servidor ao qual foi destinada aquela conexão, e a política do balanceador de carga[2];
- Um **IPS** Snort, que é um *software* de detecção/prevenção de intrusões para sistemas Linux, disponibilizado sem custos em [48]. Ele envia para o protótipo os registros de conexão e o estado da conexão perante o IPS. Esse último parâmetro assume o valor "0" quando a conexão é liberada sem alertas, o valor "1" quando ela é bloqueada, e um outro valor inteiro quando a conexão gera algum tipo de alerta.

Além disso, foi configurado um **NAT** em um arquivo de criação da topologia no Mininet. Ele não envia informações diretamente para o *host* que executa a solução de APM, mas compõe um dos ambientes de testes em que a validação do protótipo é realizada, de forma a se avaliar se o protótipo é capaz de interpretar informações de aplicação também com a presença de outros elementos intermediários. Este conjunto de dispositivos de *software* foi selecionado por estes serem alguns dos poucos dos middleboxes já disponibilizados de forma livre para SDN e com compatibilidade com as versões de OpenFlow e Open vSwitch utilizadas no Mininet, ainda que tenham sido necessárias algumas adaptações neste trabalho.

As seções a seguir detalham a implementação da base de dados em que são armazenadas as informações das conexões, o desenvolvimento das APIs por meio das quais o protótipo se comunica com os middleboxes, o modelo de obtenção/interpretação das capturas do controlador SDN, e a consolidação e disponibilização dos dados pela solução.

## 4.1 Base de Dados

A base de dados em que são armazenadas e consultadas as informações de gerenciamento de desempenho foi implementada no sistema gerenciador de banco de dados

[1] O *firewall* de camada 2 é capaz de bloquear um conjunto de endereços MAC, enquanto o *firewall* de camada 4 é capaz de bloquear também endereços IP e portas UDP/TCP.

[2] O balanceador de carga encontrado em [42] "transformava" em balanceador o primeiro *switch* da rede a se conectar, e funcionava apenas com distribuição aleatória de carga. Ele foi adaptado para "transformar" *switches* específicos em balanceadores, e para ser capaz de realizar o balanceamento de carga também na política *Round-Robin*. Nesse modo, o *pool* de servidores recebe as conexões de forma sequencial e igualitária. Por exemplo, a primeira conexão vai para o primeiro servidor, a segunda para o segundo, e assim por diante.

*open-source* PostgreSQL 9.3, disponibilizado de forma gratuita em [38]. O PostgreSQL é um sistema de bases de dados objeto-relacionais compatível com plataformas baseadas em Unix, que responde a instruções *Structured Query Language* (SQL). Ele for escolhido por ser um *software* livre e bastante comum no mercado, e que provê um bom desempenho em ambientes de larga escala, sendo inclusive utilizado por outras soluções de APM como o Riverbed SteelCentral.

Conforme indicado no Capítulo 3, na Seção 3.6, para cada aplicação cadastrada no protótipo, é criada uma tabela nessa base de dados. A tabela contém uma listagem das conexões de uma aplicação que trafegaram pela rede, e cada conexão ocupa uma linha da tabela. As colunas que representam as informações de conexão do controlador são sempre criadas, pois são obrigatórias para cada conexão. As colunas que representam as informações de middleboxes são criadas conforme a presença dessas middleboxes na rede. Por exemplo, uma rede com um *firewall* conterá uma coluna adicional indicando o estado da conexão perante o *firewall* (bloqueada/liberada). Caso houvesse dois *firewalls* na rede, seriam criadas duas colunas adicionais deste tipo. Em uma topologia com um *firewall*, um IPS e um balanceador de carga, por exemplo, seriam utilizados os comandos ilustrados no Apêndice A, Seção A.1. As colunas da tabela criada identificam os seguintes parâmetros de uma conexão:

- IPSrc: endereço IP de origem (texto);
- IPDst: endereço IP de destino (texto);
- PortSrc: porta de origem (número inteiro);
- PortDst: porta de destino (número inteiro);
- TotalBytes: total de *bytes* trafegados na conexão;
- Duration: duração total da conexão (número real, representado como o tipo *numeric*)[3];
- ConnStart: data e horário de início da conexão (texto);
- ConnEnd: data e horário de fim da conexão (texto);
- StatusFW: estado da conexão perante o *firewall*, representado com um tipo *boolean*. Assume o valor *True* se a conexão for liberada pelo *firewall*, e o valor *False* se ela for bloqueada;
- StatusIPS: estado da conexão perante o IPS, representado como um número inteiro. Assume o valor "0" se a conexão é liberada sem alertas, "1" se ela é bloqueada, e um outro valor inteiro quando é liberada gerando algum tipo de alerta;
- StatusLBPool: estado do *pool* de servidores do balanceador quando a conexão chega a ele. Assume o valor *True* se houver algum servidor ativo, e o valor *False* se não houver servidores ativos no *pool*;
- ChosenServerLB: endereço IP do servidor que atendeu à requisição (texto);

[3]O protótipo interpreta a duração da conexão como o tempo de resposta da aplicação (ignorando, portanto, o tempo de processamento no cliente).

- LBPolicy: política de balanceamento de carga (texto);
- Ready: variável do tipo *boolean* que indica se a conexão já está pronta para ser exibida ao administrador da solução de APM. Ela é inicialmente configurada com o valor *False*, e passa a ter o valor *True* quando já foi alterada pelos middleboxes (já passou pelo estágio de UPDATE descrito na Seção 3.6). Essa variável foi criada para evitar que uma conexão inserida no banco pela análise de capturas do controlador fosse exibida ao administrador com informações incompletas;
- AbsConnStart: variável numérica que indica o tempo absoluto de início da conexão, conforme identificado pela captura do controlador[4]. Optou-se por ter colunas "redundantes" de início e fim da conexão para evitar um excesso de conversões de tipos entre data absoluta e data no modo texto (*human-readable*). As variáveis de tempo absoluto tornam mais rápida e simples a comparação entre datas/horários, enquanto que as datas em texto podem ser compreendidas diretamente por um administrador. Entendeu-se que o dispêndio de armazenamento com duas colunas a mais é pouco significativo em relação ao processamento provocado por conversões frequentes;
- AbsConnEnd: variável numérica que indica o tempo absoluto de fim da conexão.

As capturas de tráfego do controlador SDN são executadas de tempos em tempos, em um intervalo configurado pelo admistrador. A cada conexão identificada em uma captura de tráfego completa e tratada, é realizado imediatamente uma instrução SQL *INSERT* na tabela de conexões, inserindo uma nova linha com os parâmetros IPSrc, IPDst, PortSrc, PortDst, TotalBytes, Duration, ConnStart, AbsConnStart e AbsConnEnd preenchidos. As demais colunas da linha permanecem com seus valores padrão. Quando essa mesma conexão atinge um middlebox, as colunas correspondentes a esse middlebox são atualizadas com uma instrução do tipo *UPDATE* na tabela, realizada com alguns segundos de atraso para garantir que o *INSERT* da conexão já tenha sido feito. Na implementação do protótipo, esse atraso é configurado como 20 segundos somados ao intervalo entre uma captura e outra. Por exemplo, se é realizada uma captura a cada 30 segundos, o *UPDATE* será feito 50 segundos após o momento em que a conexão chega ao middlebox, e portanto a conexão só será disponibilizada ao administrador (coluna $Ready = True$) 50 segundos depois de sua ocorrência. Os 20 segundos são um intervalo "extra" para garantir que o *INSERT* já terá sido executado no banco de dados. Em geral, em um sistema com maiores recursos computacionais para a solução, esse intervalo poderia ser muito menor.

Com a expectativa de expansão da solução de APM para que ela possa ser implementada em um conjunto de *hosts* ao invés de em um *host* individual, para situações em que seja necessário um maior poder de processamento, a base de dados não é executada no *host* da solução em si, mas sim em uma máquina indicada por um endereço IP externo. No Mininet, é utilizado o *namespace* de *root* para a execução do banco de dados, que é compartilhado por todos os *hosts* do Mininet. O endereço IP de "*root*" é então configurado na solução de APM como o endereço IP externo do banco. Em um ambiente de produção,

[4]Em Python, a função *time* da biblioteca *time* em Unix indica o número absoluto de segundos decorridos desde 01/01/1970 às 00:00 horas, como uma variável do tipo ponto flutuante.

esse endereço IP normalmente seria o de um banco de dados corporativo, acessível a um conjunto de servidores que compõem a solução.

As colunas AbsConnStart e AbsConnEnd permitem a realização de "limpezas" da base de dados de acordo com um período de retenção definido. Cria-se uma rotina executada periodicamente na *cron*[5] do Sistema Operacional da base de dados que faça a deleção dos dados antigos. Tomando por base o momento de término de uma conexão para a retenção, o comando na *cron* indicaria a conexão ao banco de dados e a execução de uma instrução de deleção, conforme exemplificado no Apêndice A, Seção A.2.

Para simplificar a administração do SGBD PostgreSQL, foi utilizado o *software* de administração gratuito pgAdmin3, disponível em [37]. A Figura 4.1 exibe um exemplo de tabela criada com o código exemplificado, sendo visualizada na ferramenta de administração pgAdmin3. A Figura 4.2 exibe a interface de administração do pgAdmin3, em uma conexão à base de dados dbh7 no endereço IP de gerenciamento (172.16.106.128), na porta padrão do PostgreSQL (5432).

| IPSrc text | IPDst text | PortSrc integer | PortDst integer | TotalByt integer | Duration numeric | ConnStart [PK] text | ConnEnd text | StatusFW boolean | StatusIPS integer | StatusLBP boolean | ChosenSer text | LBPolicy text | Ready boolean | AbsCo numeri | AbsCo numeri |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 192.168.2.101 | 10.0.0.10 | 51570 | 8000 | 152 | 16.1491 | 2015-07- | 2015-07- | FALSE | 0 | TRUE | 10.0.0.4 | Random | TRUE | 14363 | 14363 |
| 192.168.2.103 | 10.0.0.10 | 35644 | 8000 | 456 | 15.6531 | 2015-07- | 2015-07- | TRUE | 0 | TRUE | 10.0.0.5 | Random | TRUE | 14363 | 14363 |
| 192.168.2.102 | 10.0.0.10 | 47327 | 8000 | 152 | 0.9380 | 2015-07- | 2015-07- | TRUE | 1 | TRUE | 10.0.0.4 | Random | TRUE | 14363 | 14363 |

Figura 4.1: Exemplo de tabela do protótipo visualizada pela ferramenta pgAdmin3.

## 4.2 APIs dos Middleboxes

A presença de APIs de configuração de middleboxes de mercado voltadas para SDN é bastante comum, especialmente no caso de *softwares* proprietários, conforme indicado na Seção 3.5. Porém, como comentado anteriormente, foram selecionados apenas middleboxes gratuitos e de código aberto para este trabalho. Esses middleboxes ainda não possuíam uma API de integração, ou a possuíam de forma pouco documentada. Assim, optou-se por desenvolver um padrão de API que tornasse possível a obtenção dos parâmetros que definem o estado de cada middlebox, conforme a descrição do início deste capítulo. O padrão de API segue os critérios da modelagem descrita na Seção 3.5: a cada nova conexão identificada no middlebox, ele envia utilizando o padrão JSON os parâmetros de estado para um endereço IP que representa a solução de APM (no caso, um *host* que executa a solução, ou um *pool* de *hosts* com um endereço IP respondendo pelo *pool*).

Para o *firewall* e para o balanceador de carga, no código de cada middlebox (desenvolvido em Python), em pontos específicos de identificação de novas conexões, é inserida a função *json.dumps* para alocar os parâmetros de estado da conexão a um vetor de dados JSON (variável *data*). No IPS, é feita uma configuração semelhante, mas a partir do monitoramento dos *logs* ao invés da modificação do código do *software*, conforme será detalhado a seguir. O vetor de dados JSON já "preenchido" é enviado ao endereço IP de

[5]A *cron* é um agendador de tarefas para Sistemas Operacionais baseados em Unix, em que se configuram comandos ou *scripts* para serem executados em datas específicas ou de forma periódica.

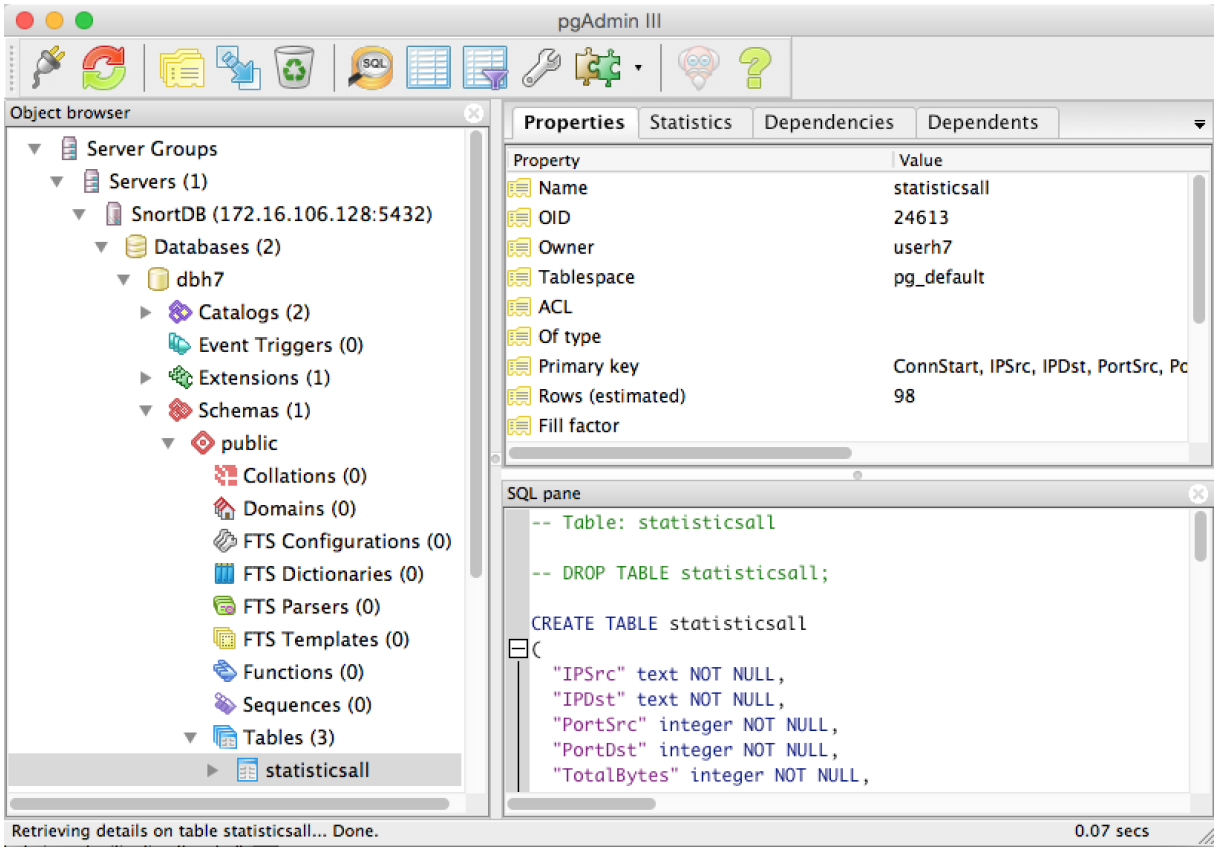



Figura 4.2: Interface de administração da base de dados do protótipo na ferramenta pgAdmin3.

monitoramento utilizando um POST HTTP [6]. O protótipo, que executa um serviço Web na porta 5000 para receber essas requisições POST HTTP, possui funções que são acionadas a cada requisição proveniente de um dado middlebox. Existem funções e caminhos (*Uniform Resource Locators*, URLs) específicos no servidor HTTP da solução de APM para cada middlebox. Um POST HTTP do IPS é direcionado à URL *http://<endereço-monitoramento>/alertips* e aciona no protótipo a função *alertips*, um POST HTTP do *firewall* é direcionado à URL *http://<endereço-monitoramento>/alertfw* e aciona no protótipo a função *alertfw*, e um POST HTTP do balanceador de carga é direcionado à URL *http://<endereço-monitoramento>/alertlb*, acionando no protótipo a função *alertlb*.

Um excerto de código da API em Python inserido em partes específicas do código do *firewall* é exibido a título de exemplo na Seção A.3 do Apêndice A. O código da API inserido no *firewall* faz com que ele envie a cada novo fluxo tratado por ele um vetor de dados JSON (variável *data*). Uma vez que o vetor JSON é enviado por POST HTTP ao servidor Web do protótipo de solução de APM, o servidor recebe cada um dos dados do vetor a partir da função *request.json*, com os mesmos nomes configurados na API do middlebox. Um exemplo de código em que o servidor de monitoramento recebe as informações do *firewall* é exibido na Seção A.4. A Seção A.5 do Apêndice A, por sua vez, exibe o código com que o serviço Web é implementado na solução de monitoramento.

[6] Um POST é uma mecanismo do protocolo HTTP que requisita que um servidor Web aceite/armazene os dados que estão no corpo da mensagem.

O serviço Web utiliza a classe Flask, que possui o padrão de comunicação *Web Server Gateway Interface* [58] [21]. O objeto flask atua como um registro central para as regras de URL e as funções correspondentes.

À medida que novos POSTs HTTP atingem a função do middlebox correspondente (por exemplo, a função *alertfw*), essa função acumula um conjunto de instruções SQL do tipo UPDATE direcionadas à tabela de conexões. Uma vez decorrido um certo atraso para garantir que as linhas correspondentes já tenham sido inseridas na tabela, esse conjunto de instruções é executado, conforme explicado na seção anterior. Além de esse atraso ter o propósito de garantir consistência nas informações das conexões, ele também evita que um número excessivo de conexões ao banco de dados seja criado, o que impactaria negativamente o seu desempenho. Por exemplo, em uma aplicação cujas conexões passam por cinco middleboxes, cada nova conexão geraria cinco conexões e UPDATEs no banco de dados caso não houvesse esse atraso/"acúmulo" dos UPDATEs, o que seria potencialmente desastroso em um ambiente com conexões muito frequentes. Assim, enquanto se aguarda o atraso configurado, as instruções vão sendo armazenadas em um arquivo texto localizado em um espaço compartilhado por todos os *hosts* da topologia, assim como a base de dados.

Apresenta-se na Seção A.6 um trecho do código da função *alertfw* (executada no protótipo para tratar as informações recebidas do *firewall*) em que a instrução de UPDATE é armazenada em um arquivo, que será lido e "reciclado" de tempos em tempos (arquivos dessa forma são criados periodicamente a cada intervalo de UPDATE). Para que as instruções UPDATE sejam executadas, define-se uma função de conexão com a base de dados e uma função de atualização dessa base, executados em Python no protótipo. Os códigos dessas funções são ilustrados na Seção A.7. A execução das funções de conexão e atualização do banco exibidas acima exige a importação da biblioteca Psycopg2, um *driver* de API para interação de programas em Python com bases de dados no SGBD PostgreSQL [57].

A única API um pouco diferente das demais é a API do IPS, por trabalhar a partir dos seus *logs* ao invés de exigir modificações de código. Acredita-se que na maioria dos middleboxes de mercado é possível seguir uma abordagem deste tipo caso já não exista uma API implementada, pois os parâmetros de conexão requisitados já estão presentes em boa parte dos *logs* de dispositivos que têm qualquer atuação sobre o tráfego de rede. O código da API do IPS Snort é exibido na Seção A.8 do Apêndice A, para que sirva como base para futuros desenvolvimentos de APIs para a solução de APM proposta.

## 4.3 Capturas do Controlador

As capturas do tráfego de rede que passa pelo controlador são realizadas utilizando o utilitário *dumpcap*, e assim como a base de dados e os arquivos de UPDATE dos middleboxes, elas são armazenadas em um espaço em disco compartilhado por todos os *hosts*. O *dumpcap* é uma ferramenta de captura de pacotes semelhante ao *software* Wireshark, mas que possui uma execução mais leve e interface apenas em linha de comando [16]. O próprio controlador executa o *dumpcap* periodicamente, com controle por meio de um *script* em Python. O *script* é executado quando o administrador inicia o protótipo de gerenciamento de desempenho, da seguinte forma:

```
python receive-capture-all.py -d <Service_IP> -p <Service_Port>
    -i <Interval> -t <Total_time_in_seconds> -h <
    Monitoring_host_IP>
```

São passados para o início do *script* os seguintes parâmetros:

1. *Service_IP*: o endereço IP do servidor da aplicação. O *script* pode ser facilmente ajustado para receber vários IPs de servidores de uma mesma aplicação. Caso se deseje monitorar múltiplas aplicações, seriam iniciados vários *scripts* de captura, cada um passando os parâmetros de uma aplicação;

2. *Service_Port*: a porta da aplicação. O *script* assume que cada aplicação funciona em uma porta específica, mas pode ser facilmente ajustado para considerar um conjunto de portas;

3. *Interval*: o intervalo entre a execução de uma captura e a seguinte, em segundos, definido pelo administrador. A cada arquivo de captura completo, é realizado um INSERT das informações de conexão na base de dados, e alguns segundos depois, o UPDATE com as informações dos middleboxes, deixando a conexão pronta para ser disponibilizada pro administrador da solução. Assim, esse valor de intervalo será o principal parâmetro para determinar qual a frequência possível de disponibilização dos dados finais da solução de APM. Portanto, ele não deve ser muito grande, para que os dados das conexões sejam disponibilizados em um tempo próximo ao tempo real de sua ocorrência, e para que as métricas de disponibilidade e tempo de resposta das aplicações também não sejam muito defasadas. Ao mesmo tempo, um intervalo muito curto incorreria em um consumo de recursos de processamento e de I/O (*input/output* de disco) muito maior pela solução de APM, principalmente por causa da função de análise executada em seguida sobre cada captura. De forma geral, considera-se que um intervalo entre 1 minuto e 5 minutos seria razoável, a depender dos recursos disponíveis e da precisão de tempo desejados para a solução;

4. *Total_time_in_seconds*: o tempo total em segundos de execução de todas as capturas. Esse parâmetro possibilita estipular um tempo limite para o funcionamento do protótipo, que pode ser "infinito" (muito grande) para que a solução seja executada indefinidamente;

5. *Monitoring_host_IP*: o endereço IP da solução de APM.

Uma vez passados esses parâmetros ao *script*, ele irá executar capturas com a periodicidade do *Interval* no controlador com o *dumpcap*, até atingir o tempo de execução definido pelo *Total_time_in_seconds*. Cada comando *dumpcap* é executado no controlador da seguinte forma:

```
sudo dumpcap -i any -a duration:<Interval> -f "host" <Service_IP
    > and tcp port <Service_Port> -w <Arquivo_de_Saida.pcapng>
```

Quando uma captura é concluída, um componente em Python executado no servidor da solução de gerenciamento chama uma função de análise, cujo código é exibido na Seção A.9. Essa função primeiramente "recicla" o arquivo de estatísticas de middleboxes, ou seja, move o arquivo atual para um arquivo histórico, para que sejam criados arquivos de estatísticas de middleboxes com a mesma periodicidade do intervalo de captura. A

partir desse momento, os 20 segundos de atraso entre a captura de pacotes e a atualização dos middleboxes começam a ser contados. Em seguida, a função utiliza a ferramenta Tshark para gerar as estatísticas de todas as conexões daquela captura. O Tshark é uma versão do *software* de análise Wireshark em linha de comando [52], e com as opções "*-q, -v conv,tcp*", ele gera as mesmas estatísticas de conexões TCP que seriam acessadas no Wireshark na aba *Statistics* → *Conversations* → *TCP*. Essas estatísticas são armazenadas em um arquivo de extensão ".csv", que é armazenado também em um espaço em disco compartilhado por todos os *hosts* da topologia. Elas são exemplificadas na Figura 4.3, da forma como são exibidas na interface gráfica do Wireshark.

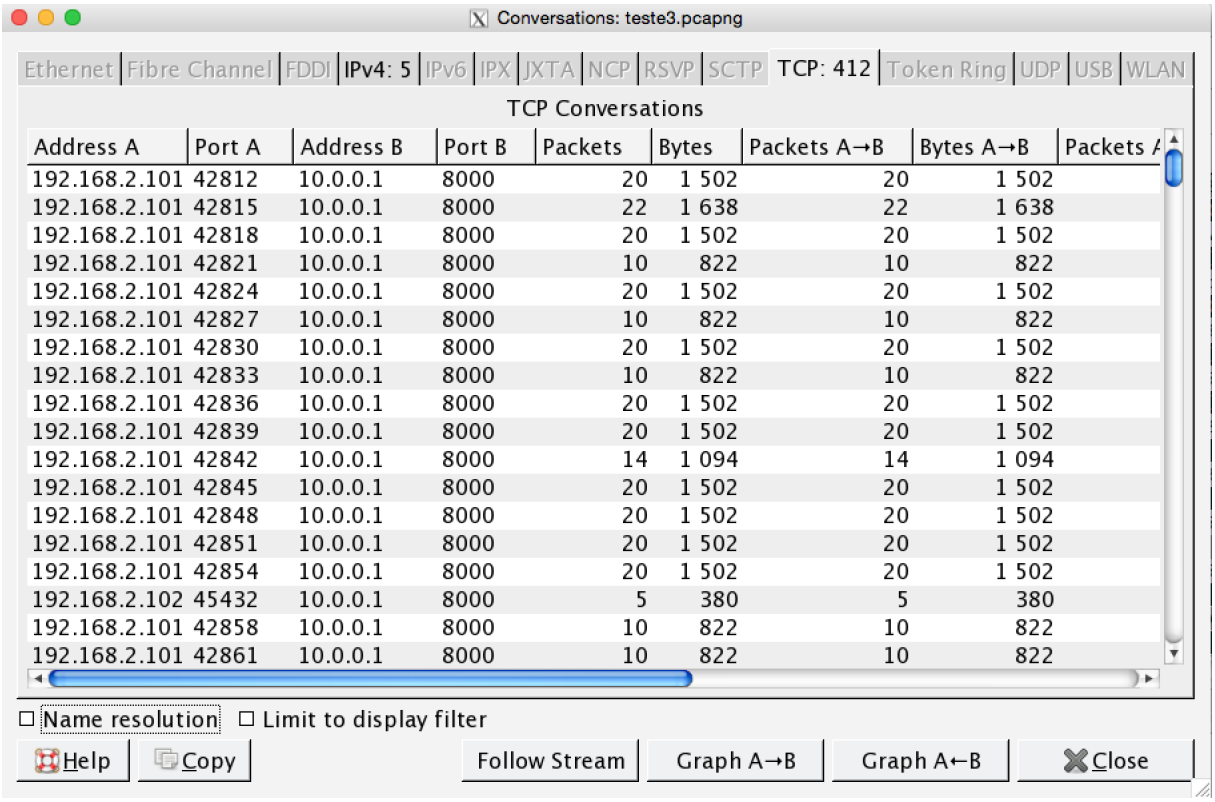



Figura 4.3: Exemplo de estatísticas de conexões TCP no Wireshark. O Tshark gera uma saída equivalente em texto, armazenada um arquivo ".csv".

Alguns tratamentos adicionais são feitos sobre os resultados do Tshark, de forma a preparar o arquivo de saída para a análise pela próxima função, que irá executar as instruções de INSERT das estatísticas de conexão na base de dados. A função seguinte, que é executada pelo servidor da solução de APM, trata as informações dos arquivos ".csv" e executa as instruções de INSERT na base de dados. Assim, as informações de cada conexão, de acordo com o Tshark e com alguns tratamentos adicionais, são armazenadas na base de dados, com cada conexão ocupando uma linha da tabela. O código dessa função é exibido de forma resumida na Seção A.10. Para cada novo grupo de informações, é utilizada a função *insert_DB* para inseri-lo na base de dados com a instrução INSERT. O código dessa função é exibido na Seção A.11.

A Figura 4.4 resume a forma como foi implementado o protótipo, indicando os principais componentes citados.

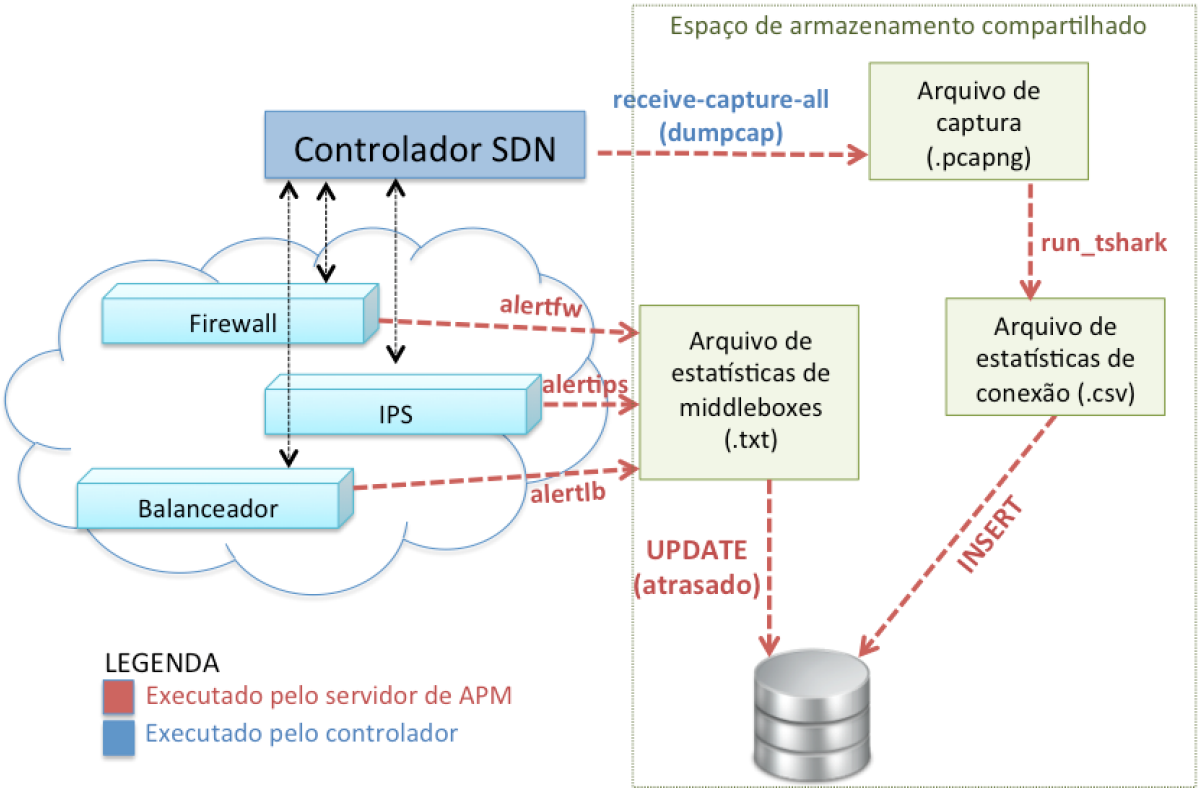



Figura 4.4: Mecanismo de funcionamento do protótipo.

## 4.4 Disponibilização dos Dados

A disponibilização das informações da solução para o administrador da solução é realizada a partir de uma página Web que monitora continuamente a base de dados. São executadas instruções de SELECT na base para buscar e tratar as conexões mais recentes, assumindo que cada aplicação possua uma tabela própria. O administrador informa para o programa a base de dados e a tabela onde estão as informações da aplicação, um período de atualização e a quantidade de conexões a serem exibidas. A página Web irá exibir informações das conexões mais recentes (limitadas à quantidade especificada), bem como a disponibilidade e o tempo de resposta da aplicação para esse conjunto de conexões. Em sua configuração padrão, a página exibe informações relativas às 10 conexões mais recentes, e é atualizada a cada minuto.

A Figura 4.5 ilustra a interface Web do protótipo. Essa interface foi desenvolvida utilizando as linguagens HTML[7], CSS[8], PHP[9] e JavaScript[10], a partir de *templates* Bootstrap disponibilizados de forma gratuita em [7]. A página Web é executada sobre um servidor *open-source* Apache [4]. Ela exibe duas seções para o usuário administrador: **Applica-**

[7] *HyperText Markup Language* (HTML) é uma linguagem de marcação de textos para exibição em páginas Web.

[8] *Cascading Style Sheets* (CSS) é uma linguagem de definição de estilos e formatações para páginas Web.

[9] *Hypertext Preprocessor* (PHP) é uma linguagem de *scripting* que pode ser inserida em códigos HTML.

[10] JavaScript é uma linguagem de programação dinâmica e interpretada, muito utilizada em serviços Web. Neste trabalho foi utilizada principalmente a biblioteca jQuery.

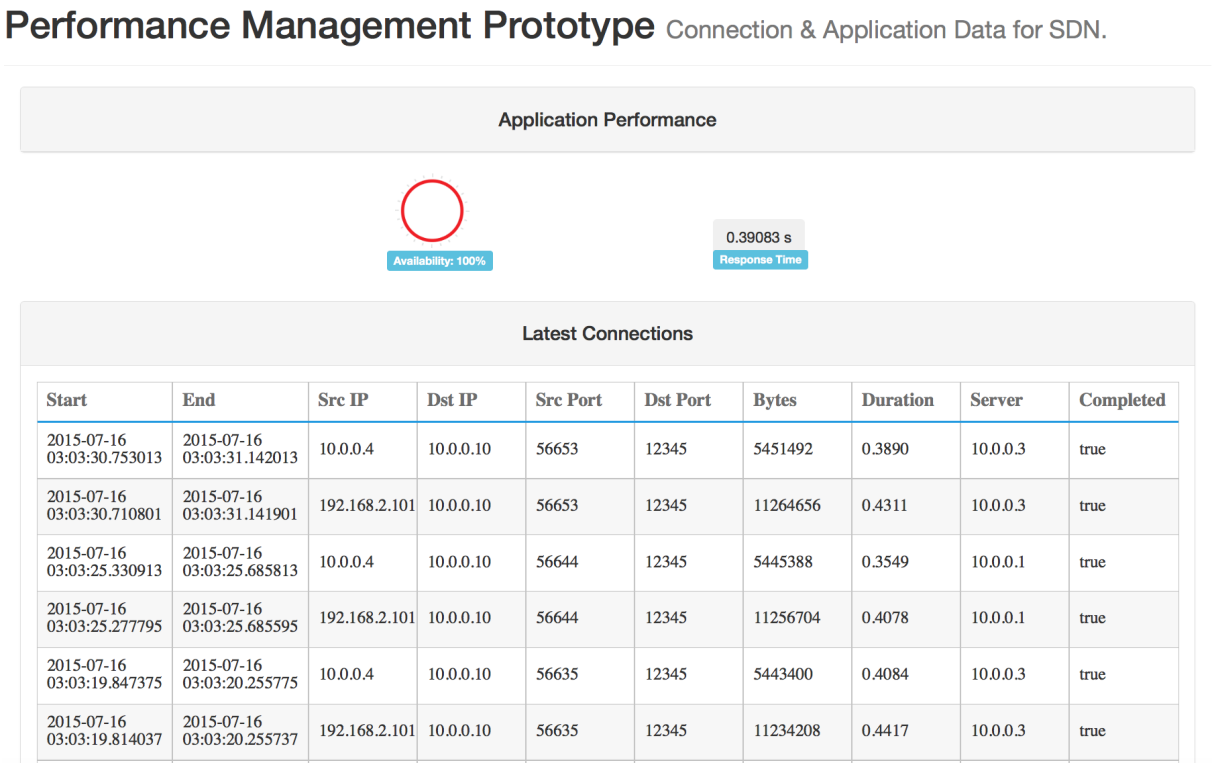



Figura 4.5: Interface Web de gerenciamento de desempenho.

**tion Performance**, com os dados de tempo de resposta e de disponibilidade da aplicação para o conjunto de conexões selecionado, e **Latest Connections**, com informações das conexões mais recentes. A Figura 4.6 exibe de forma mais próxima a seção **Application Performance**, ilustrando o valor percentual da disponibilidade e a imagem que a representa logo acima (círculo que se preenche conforme a porcentagem), bem como o valor do tempo de resposta médio da aplicação de forma destacada. A disponibilidade é computada como uma proporção das conexões bem-sucedidas, ou seja, que passam por todos os middleboxes sem nenhum bloqueio e com pelo menos um dos servidores disponíveis. O tempo de resposta é disponibilizado a partir das colunas de duração das conexões.


Figura 4.6: Seção "Application Performance" da interface Web do protótipo.

A seção **Latest Connections** exibe as seguintes informações das conexões mais recentes: horário de início (**Start**), horário de término (**End**), endereço IP de origem (**Src IP**), endereço IP de destino (**Dst IP**), porta de origem (**Src Port**), porta de destino (**Dst Port**), quantidade de *bytes* trafegados (**Bytes**), duração da conexão (**Duration**),

servidor que atendeu à conexão (**Server**) e a indicação de que a conexão foi completada (bem-sucedida) ou não (**Completed**). Essa última informação é obtida a partir do conjunto de colunas da tabela que indicam o estado da conexão perante os múltiplos middleboxes (caso algum deles indique que a conexão não foi bem-sucedida, esse campo muda seu valor para "false"). Optou-se por não exibir todas as informações de conexão na interface Web para simplificar a visualização para o administrador, trazendo maior foco às informações mais relevantes e deixando a interface mais agradável visualmente.

Caso exista a necessidade de monitorar as conexões e seus parâmetros completos em um período específico, é disponibilizado um programa em Python que exibe em modo texto (linha de comando) essas informações. Ele recebe em sua inicialização um intervalo de atualização e o período a ser monitorado. Por exemplo, o administrador pode selecionar a visualização das conexões dos últimos 15 minutos, com atualização a cada 30 segundos. É possível ajustar esse programa de outras formas, como para ordenar as conexões em duração decrescente, ou para exibir as últimas N conexões, por exemplo. Para cada conexão, em uma topologia com um middlebox de cada tipo dentre os que podem ser interpretados, o programa em Python exibe os seguintes parâmetros: horário de início (**Start**), horário de término (**End**), endereço IP de origem (**Src IP**), porta de origem (**Src Port**), porta de destino (**Dst Port**), quantidade de *bytes* transferidos (**Bytes**), duração (**Duration**), estado perante o *firewall* (**Status FW**), estado perante o IPS (**Status IPS**), estado do *pool* de servidores no balanceador de carga (**Status LB Pool**) e política do balanceador (**LB Policy**).

## 4.5 Considerações de Escalabilidade

O protótipo foi validado em topologias distintas, como será explicado no Capítulo 5. Foram realizadas avaliações com mais de uma aplicação, mais de um controlador SDN e com um conjunto de middleboxes atuando simultaneamente. Em todas essas topologias foram obtidos bons resultados. O protótipo, da forma como foi desenvolvido, deverá funcionar com qualquer número de controladores SDN, desde que cada controlador execute o código que gera as capturas e as armazene com nomes diferentes. O número de middleboxes com que o protótipo pode funcionar também é ilimitado, bastando criar colunas repetidas e com nomes diferentes nas tabelas de conexões para middleboxes de um mesmo tipo (por exemplo, coluna "Status IPS 1" e coluna "Status IPS 2"), e fazer um ajuste correspondente para que o programa que faz as atualizações de tabelas possa interpretar essas colunas "duplicadas". Da mesma forma, a quantidade de aplicações que o protótipo suporta também é virtualmente ilimitada, devendo-se apenas criar uma tabela separada no banco de dados para cada aplicação, indicando os nomes das tabelas para os módulos do sistema.

Contudo, todas as topologias em que o protótipo foi validado continham um número limitado de *hosts* clientes e servidores. O desempenho do protótipo da solução de APM naturalmente irá piorar caso o número de conexões, middleboxes, controladores e *hosts* seja muito grande. Uma das maiores limitações de escalabilidade do protótipo de APM é a execução do Tshark para a análise de capturas, que pode demandar uma quantidade de recursos razoável e levar um tempo considerável em sua execução se a captura for muito grande. Outra limitação presente no protótipo é que atualmente ele é executado em apenas um servidor. Ainda que as instruções realizadas no SGBD sejam executadas

em um tempo muito rápido (são apenas instruções de SELECT, INSERT e UPDATE, que em geral não demandam muito da base de dados), o processamento das funções que geram essas instruções pode ser oneroso dependendo da quantidade de conexões.

Uma proposta de solução para adaptar o protótipo a um modelo mais escalável é distribuir a execução das funções de verificação de capturas e de estados de middleboxes entre vários servidores. Criaria-se um *pool* de servidores da solução de APM, com um *proxy gateway* expondo o seu endereço IP para os controladores e os middleboxes, e distribuindo os arquivos de capturas ".pcapng" para processamento por diversos servidores com a carga aproximadamente balanceada entre eles. Esse balanceamento poderia ser feito por um mecanismo de simples implementação, como o *Round Robin* ou a distribuição aleatória entre os servidores. Esse *proxy gateway* também interceptaria os POSTs HTTP dos middleboxes e os redirecionaria para os servidores do *pool*. Assim, a maior parte das limitações de escalabilidade do protótipo estariam solucionadas. O *framework* Pyro [1] também pode ser uma alternativa interessante para distribuir as tarefas entre um *pool* de *hosts*, permitindo que objetos de diferentes *scripts* Python se comuniquem, e viabilizando alternativas de implementação do *pool* como um *cluster* de *High Performance Computing* (HPC).

Para solucionar a limitação imposta pelo tempo e pelos recursos computacionais demandados pelo Tshark em capturas muito grandes (que iria existir ainda que existissem vários servidores compondo a solução), pode-se fazer um *parse* das capturas à medida que elas são realizadas. O *parse* pode ser feito no próprio código em Python de obtenção das capturas, utilizando uma biblioteca como a Pypcap [39]. O *parse* interpretaria as conexões logo que fossem detectadas na captura, já escrevendo nos arquivos ".csv" as estatísticas de cada conexão. No entanto, para essa adaptação, seria necessário desenvolver o código para interpretar os pacotes e deduzir as estatísticas que seriam escritas no arquivo. Essa adaptação, bem como o ajuste da solução para trabalhar com vários servidores com um *proxy gateway* fazendo a interface com controladores e middleboxes, são propostos como trabalhos futuros.

# Capítulo 5

# Validação do Protótipo

Este capítulo discute as avaliações realizadas para a validação do protótipo de monitoramento de desempenho. Primeiramente, são realizadas avaliações parciais do protótipo, com apenas um tipo de aplicação (um serviço Web) e apenas um middlebox enviando informações para a solução de APM. Depois, são realizadas algumas avaliações finais, com dois tipos de aplicações (um serviço Web e um compartilhamento de arquivos) e quatro middleboxes presentes na topologia: um IPS, um NAT, um balanceador de carga e um *firewall*. No último teste, a topologia avaliada possui dois controladores de rede SDN.

Em todas as avaliações, foram configurados *scripts* sequenciais de forma a verificar se a solução de APM é executada corretamente e disponibiliza os parâmetros de monitoramento corretos. São parâmetros de monitoramento avaliados em todos os testes: o tempo de resposta e a disponibilidade da aplicação monitorada, e para cada conexão, a quantidade de *bytes* trafegados, os endereços IP de origem e destino, as portas de origem e destino, os horários de início e fim e a duração. Na presença do *firewall*, é avaliado também o estado da conexão perante o *firewall*; na presença do IPS, é avaliado o estado da conexão perante o IPS; na presença do balanceador, são avaliados também para cada conexão o endereço IP do servidor que atendeu à requisição, o estado do *pool* de servidores e a política de balanceamento.

## 5.1 Ambiente Experimental

O ambiente de testes foi configurado em uma máquina virtual do Mininet 2.1.0, de Sistema Operacional Ubuntu 14.04 LTS, com 2 CPUs virtuais e 8 GB de memória RAM. Nessa máquina se emulavam diferentes topologias com *switches* Open vSwitch 1.3 e com o controlador POX 0.3.0. Nela, foram instalados alguns utilitários adicionais:

- HTTPing 1.5.8 para realizar requisições HTTP e medir seus tempos de resposta [29];
- CGIHTTPServer, um servidor Web em Python capaz de lidar com requisições HTTP do tipo *Common Gateway Interface* (CGI) [13];
- Dumpcap 1.5.3 para captura de pacotes;
- Wireshark e Tshark 1.10.6 para captura e análise de pacotes;
- IPS Snort 2.9.7.0.

## 5.2 Avaliações Parciais do Protótipo

Foram realizadas duas avaliações parciais do protótipo, cada uma com três testes, em que a aplicação avaliada é um serviço Web. Na primeira avaliação, o funcionamento do protótipo é testado com um único middlebox, um balanceador de carga. Na segunda, ele é avaliado em uma topologia com dois middleboxes, um IPS e um NAT, mas apenas com o IPS interagindo continuamente com a solução de APM. Em ambos os testes, um ou mais *hosts* da topologia executam um servidor Web CGIHTTPServer, e um ou mais *hosts* atuam como clientes dessa aplicação Web, fazendo requisições sequenciais com o HTTPing. São executados *scripts* em que são definidas as requisições HTTP realizadas por cada host, e os momentos em que um determinado link ou dispositivo fica disponível/indisponível, de forma a variar a disponibilidade da aplicação.

Para cada uma das duas avaliações, os três testes realizados (T1, T2 e T3) são descritos resumidamente na Tabela 5.1. A primeira avaliação ("Testes com o balanceador de carga") utiliza a topologia indicada na Figura 5.1. A segunda avaliação ("Testes com o IPS") utiliza a topologia da Figura 5.2. Para validação dos resultados, as métricas obtidas pelo protótipo são comparadas a um conjunto de informações que serão denominadas **dados de validação**:

- O próprio *script* de testes para o controle da disponibilidade e dos parâmetros dos middleboxes (sabe-se exatamente em que ponto do *script* há falhas planejadas em qualquer dos componentes ou mudanças de configuração nos middleboxes);
- Capturas nas interfaces do ambiente emulado no Mininet utilizando o Wireshark para validar os dados de conexão obtidos pelas análises de capturas do controlador SDN;
- Os resultados do HTTPing para verificar o tempo de resposta de cada requisição.

Espera-se, de forma geral, que os resultados do protótipo sejam iguais aos dados de validação. No entanto, poderão existir pequenas diferenças entre os tempos de resposta do protótipo e os dados de validação, visto que a medição realizada pelo HTTPing é um pouco diferente da realizada pelo protótipo. O protótipo considera o tempo de resposta como o intervalo de tempo desde a saída do primeiro pacote até o recebimento do último pacote de uma conexão (camada 4), e o HTTPing considera o tempo desde a requisição pela ferramenta até o recebimento da resposta completa (camada 7). Além disso, o HTTPing trunca os tempos de resposta em duas casas decimais, enquanto que o protótipo mantém quatro casas. Assim, define-se um **valor de referência** máximo aceitável para essas diferenças entre os tempos de respostas, de forma a validar se o resultado do teste é aceitável ou não. Para os testes a seguir, será considerado um valor de referência de **10%** do tempo de resposta médio de validação (HTTPing). Ou seja, os resultados de tempo de resposta do protótipo serão considerados válidos caso as diferenças sejam menores do que esse valor de referência.

### 5.2.1 Testes com o Balanceador de Carga

Para os três testes com o balanceador, foi emulada a topologia ilustrada na Figura 5.1. Um exemplo de parâmetros de saída desse teste para uma conexão é exibido na Tabela

Tabela 5.1: Avaliações parciais realizadas para validar o protótipo.

| | Balanceador | IPS |
|---|---|---|
| T1 | Cada *host* cliente faz 100 requisições HEAD. Não há indisponibilidades. | Cada *host* cliente faz 100 requisições head. Ambiente disponível e sem bloqueios. |
| T2 | Cada *host* cliente faz 100 requisições GET de um arquivo de 3900 *bytes*. Não há indisponibilidades. | Cada *host* cliente faz 100 requisições GET de um arquivo de 3900 *bytes*. Ambiente disponível e sem bloqueios. |
| T3 | Cada *host* cliente faz 300 requisições GET de um arquivo de 3900 *bytes*. Servidores são indisponibilizados na sequência h4 → h5 → h6, e retornam em sequência igual. | Cada *host* cliente faz 100 requisições HEAD. Conexões dos *hosts* h1 e h2 são bloqueadas pelo IPS (serviço indisponível para h1 e h2). Restante do ambiente disponível. |

5.2. As Tabelas 5.3, 5.4 e 5.5 exibem os resultados de tempo de resposta médio e de disponibilidade média para as conexões de cada *host* cliente (h1, h2 e h3) e para o total de conexões, em comparação com os dados de validação[2].

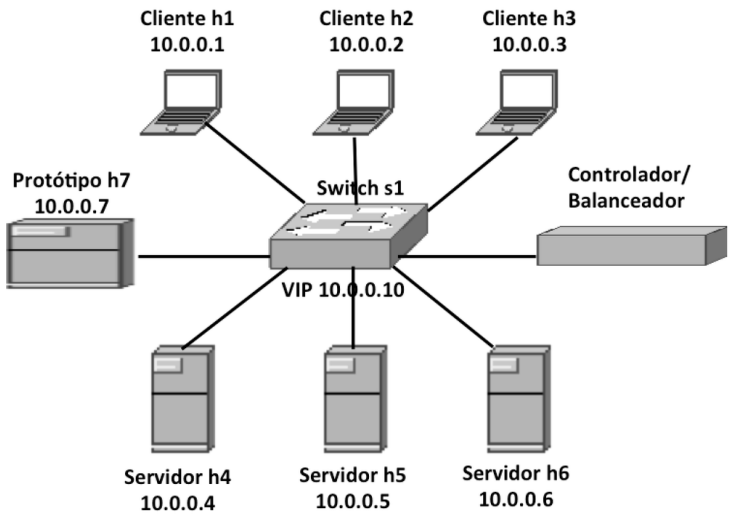



Figura 5.1: Topologia utilizada nos testes inicias com o balanceador de carga.

As informações de disponibilidade do protótipo corresponderam exatamente às dos dados de validação. Essa disponibilidade era verificada a partir da API do balanceador de carga, que checava antes de encaminhar cada requisição HTTP se havia servidores disponíveis para atendê-la, e informava esse estado ao *host* que executava a solução de APM (h7). Ela foi calculada como a porcentagem de requisições que foram atendidas para cada *host*. Apenas no teste 3, em que houve uma interrupção sequencial dos links de comunicação dos *hosts* h4, h5 e h6, foi notada alguma indisponibilidade. O período

[2]No teste 3 foram desconsideradas as requisições HTTP que não foram respondidas para o cômputo do tempo de resposta. Portanto, o cálculo do tempo de resposta conta com um número menor de conexões de cada *host*.

Tabela 5.2: Exemplo de métricas obtidas no protótipo para uma conexão realizada a partir do *host* h3 em um teste com o balanceador de carga.

| Métrica | Resultado |
|---|---|
| Horário | 13/03/2015 13:07:05 |
| IP origem | 10.0.0.3 |
| IP destino | 10.0.0.10 |
| Porta origem | 54730 |
| Porta destino | 8000 |
| Servidor | 10.0.0.4 |
| *Bytes* | 980 |
| Estado do *Pool* | Disponível |
| Política | Aleatória |
| Tempo Resposta Conexão | 12,23 ms |
| Tempo Resposta Médio | 18,42 ms |
| Disponibilidade Média | 99,98% |

em que o serviço ficou totalmente indisponível fez com que 4 das 300 requisições de cada *host* cliente não fossem atendidas, levando a um percentual de disponibilidade de aproximadamente 98,67%.

Já na comparação entre os tempos de resposta obtidos pelo protótipo e pelo HTT-Ping, houve diferenças sutis, descritas na linha "Diferença T. Resp." das tabelas de resultados. Conforme descrito anteriormente, pequenas diferenças nesses valores eram esperadas, desde que os valores dos tempos de resposta fossem inferiores aos valores de referência. As diferenças entre os tempos de resposta foram mínimas: no teste 1, representaram um valor médio absoluto de 0,20 ms; no teste 2, de 0,24 ms; e no teste 3, de 0,25 ms. Esses valores, bem como os de desvio padrão das diferenças, representam menos de 0,8% dos valores médios de tempo de resposta, e portanto são menores que os 10% considerados para os valores de referência. Assim, os tempos de resposta obtidos pelo protótipo podem ser considerados válidos.

Os demais dados relativos a cada conexão (IPs e portas de origem/destino, quantidade de *bytes* trocados, IP do servidor que atendeu à requisição, estado do *pool* de servidores e política de balanceamento) foram sempre idênticos aos dados de validação. Logo, considera-se que os testes com o balanceador de carga foram bem-sucedidos, pois mostraram que os dados de monitoramento trazidos pelo protótipo são consistentes. Adicionalmente, não se notou *overhead* significativo pelo uso da solução de APM.

### 5.2.2 Testes com o IPS

Para os três testes com o IPS, foi utilizada a topologia da Figura 5.2, com cinco *hosts* clientes (h1, h2, h3, h4 e h5), e um *host* servidor Web (h6). No *host* representado como IPS, é executado também um NAT entre os endereços IP 192.168.2.1 e 10.0.0.2. As Tabelas 5.6, 5.7 e 5.8 exibem os resultados de tempo de resposta médio e de disponibilidade média para o conjunto de conexões dos testes 1, 2 e 3, respectivamente.

Tabela 5.3: Principais resultados do teste T1 com o balanceador.

| | | Host 1 | Host 2 | Host 3 | Total |
|---|---|---|---|---|---|
| Protótipo | Tempo Resposta (ms) | 33,85 | 30,30 | 34,59 | 32,91 |
| | Disponibilidade (%) | 100 | 100 | 100 | 100 |
| Dados de Validação | Tempo Resposta (ms) | 33,61 | 30,22 | 34,43 | 32,76 |
| | Disponibilidade (%) | 100 | 100 | 100 | 100 |
| Diferença T. Resp. | Média Absoluta (ms) | 0,24 | 0,16 | 0,20 | 0,20 |
| | Desvio Padrão (ms) | 0,16 | 0,12 | 0,20 | 0,20 |
| | Valor de Referência (ms) | 3,36 | 3,02 | 3,44 | 3,28 |

Tabela 5.4: Principais resultados do teste T2 com o balanceador.

| | | Host 1 | Host 2 | Host 3 | Total |
|---|---|---|---|---|---|
| Protótipo | Tempo Resposta (ms) | 62,23 | 58,74 | 60,03 | 60,33 |
| | Disponibilidade (%) | 100 | 100 | 100 | 100 |
| Dados de Validação | Tempo Resposta (ms) | 62,03 | 58,63 | 59,86 | 60,17 |
| | Disponibilidade (%) | 100 | 100 | 100 | 100 |
| Diferença T. Resp. | Média Absoluta (ms) | 0,25 | 0,23 | 0,24 | 0,24 |
| | Desvio Padrão (ms) | 0,44 | 0,37 | 0,32 | 0,38 |
| | Valor de Referência (ms) | 6,20 | 5,86 | 5,99 | 6,02 |

Tabela 5.5: Principais resultados do teste T3 com o balanceador.

| | | Host 1 | Host 2 | Host 3 | Total |
|---|---|---|---|---|---|
| Protótipo | Tempo Resposta (ms) | 62,00 | 60,34 | 61,31 | 61,22 |
| | Disponibilidade (%) | 98,67 | 98,67 | 98,67 | 98,67 |
| Dados de Validação | Tempo Resposta (ms) | 61,71 | 60,11 | 61,11 | 60,98 |
| | Disponibilidade (%) | 98,67 | 98,67 | 98,67 | 98,67 |
| Diferença T. Resp. | Média Absoluta (ms) | 0,30 | 0,23 | 0,22 | 0,25 |
| | Desvio Padrão (ms) | 0,76 | 0,16 | 0,22 | 0,47 |
| | Valor de Referência (ms) | 6,17 | 6,01 | 6,11 | 6,10 |

Tabela 5.6: Principais resultados do teste T1 com o IPS.

| | | Host 1 | Host 2 | Host 3 | Host 4 | Host 5 | Total |
|---|---|---|---|---|---|---|---|
| Protótipo | Tempo Resposta (ms) | 29,84 | 30,60 | 30,50 | 33,39 | 32,55 | 31,38 |
| | Disponibilidade (%) | 100 | 100 | 100 | 100 | 100 | 100 |
| Validação | Tempo Resposta (ms) | 29,90 | 30,68 | 30,58 | 33,46 | 32,61 | 31,44 |
| | Disponibilidade (%) | 100 | 100 | 100 | 100 | 100 | 100 |
| Diferença | Média Absoluta (ms) | 0,08 | 0,09 | 0,10 | 0,08 | 0,09 | 0,09 |
| Tempo Resposta | Desvio Padrão (ms) | 0,06 | 0,07 | 0,18 | 0,10 | 0,11 | 0,11 |
| | Valor de Referência (ms) | 2,99 | 3,07 | 3,06 | 3,35 | 3,26 | 3,14 |

Tabela 5.7: Principais resultados do teste T2 com o IPS.

| | | Host 1 | Host 2 | Host 3 | Host 4 | Host 5 | Total |
|---|---|---|---|---|---|---|---|
| Protótipo | Tempo Resposta (ms) | 33,70 | 32,20 | 32,03 | 34,20 | 28,96 | 32,22 |
| | Disponibilidade (%) | 100 | 100 | 100 | 100 | 100 | 100 |
| Validação | Tempo Resposta (ms) | 33,47 | 31,92 | 31,83 | 34,08 | 28,76 | 32,01 |
| | Disponibilidade (%) | 100 | 100 | 100 | 100 | 100 | 100 |
| Diferença | Média Absoluta (ms) | 0,28 | 0,31 | 0,24 | 0,18 | 0,23 | 0,25 |
| Tempo Resposta | Desvio Padrão (ms) | 0,22 | 0,22 | 0,18 | 0,20 | 0,19 | 0,21 |
| | Valor de Referência (ms) | 3,35 | 3,19 | 3,18 | 3,41 | 2,88 | 3,20 |

Tabela 5.8: Principais resultados do teste T3 com o IPS.

| | | Host 1 | Host 2 | Host 3 | Host 4 | Host 5 | Total |
|---|---|---|---|---|---|---|---|
| Protótipo | Tempo Resposta (ms) | – | – | 29,44 | 29,78 | 29,34 | 29,52 |
| | Disponibilidade (%) | 0 | 0 | 100 | 100 | 100 | 60 |
| Validação | Tempo Resposta (ms) | – | – | 29,47 | 29,81 | 29,39 | 29,55 |
| | Disponibilidade (%) | 0 | 0 | 100 | 100 | 100 | 60 |
| Diferença | Média Absoluta (ms) | – | – | 0,06 | 0,06 | 0,06 | 0,06 |
| Tempo Resposta | Desvio Padrão (ms) | – | – | 0,04 | 0,04 | 0,4 | 0,04 |
| | Valor de Referência (ms) | – | – | 2,95 | 2,98 | 2,94 | 2,96 |

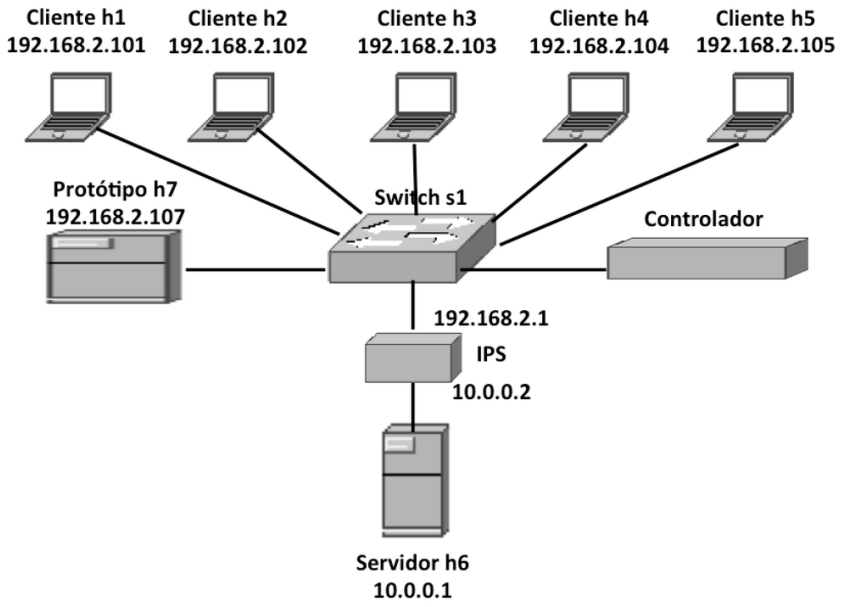



Figura 5.2: Topologia utilizada nos testes iniciais com o IPS.

Os resultados dos testes com o IPS foram bastante semelhantes aos do balanceador. As informações de disponibilidade do protótipo foram idênticas às dos dados de validação: de 100% nos testes 1 e 2, e de 60% no teste 3, pois dois dos cinco *hosts* clientes tinham suas requisições sempre bloqueadas pelo IPS. Também houve diferenças sutis nos tempos de resposta do protótipo e os obtidos pelo HTTPing: no teste 1, a diferença média absoluta foi de 0,09; no teste 2, de 0,25; e no teste 3, de 0,06. Esses valores, bem como os de desvio padrão das diferenças, representam menos de 0,8% dos valores médios de tempo de resposta e são muito inferiores aos valores de referência. Portanto, os tempos de resposta obtidos pelo protótipo podem ser considerados válidos. Os demais dados obtidos pelo protótipo (endereços IP e portas de origem/destino, quantidade de *bytes* trocados, *status* do serviço, *status* de bloqueio do IPS) foram sempre idênticos aos de validação. Assim, novamente o protótipo trouxe informações consistentes.

## 5.3 Avaliações Finais do Protótipo

Foram realizadas três avaliações finais do protótipo, para as quais foi configurada uma topologia com um IPS, um NAT, um *firewall* e um balanceador de carga. Ela contém três *hosts* clientes, h1, h2 e h3; um *pool* de servidores, composto pelos *hosts* h4, h5 e h6; e dois *switches*, um realizando o encaminhamento de pacotes em camada 2 (*switch* s2) e outro fazendo o balanceamento de carga entre o *pool* de servidores (*switch* s1). A Figura 5.3 ilustra essa topologia. O *firewall* na prática é implementado como uma funcionalidade do *switch* s1 a partir de um controlador SDN, mas é representado logicamente na Figura 5.3 como uma barreira entre os *hosts* clientes e os servidores. O dispositivo que atua como IPS também realiza NAT entre os endereços 192.168.2.1 e 10.0.0.4.

Na primeira avaliação, os servidores atuam como um servidor Web CGIHTTPServer. Na segunda avaliação, os servidores h4, h5 e h6 atuam como servidores de arquivos, enviando arquivos periodicamente para os *hosts* clientes h1, h2 e h3. Em ambos os casos,

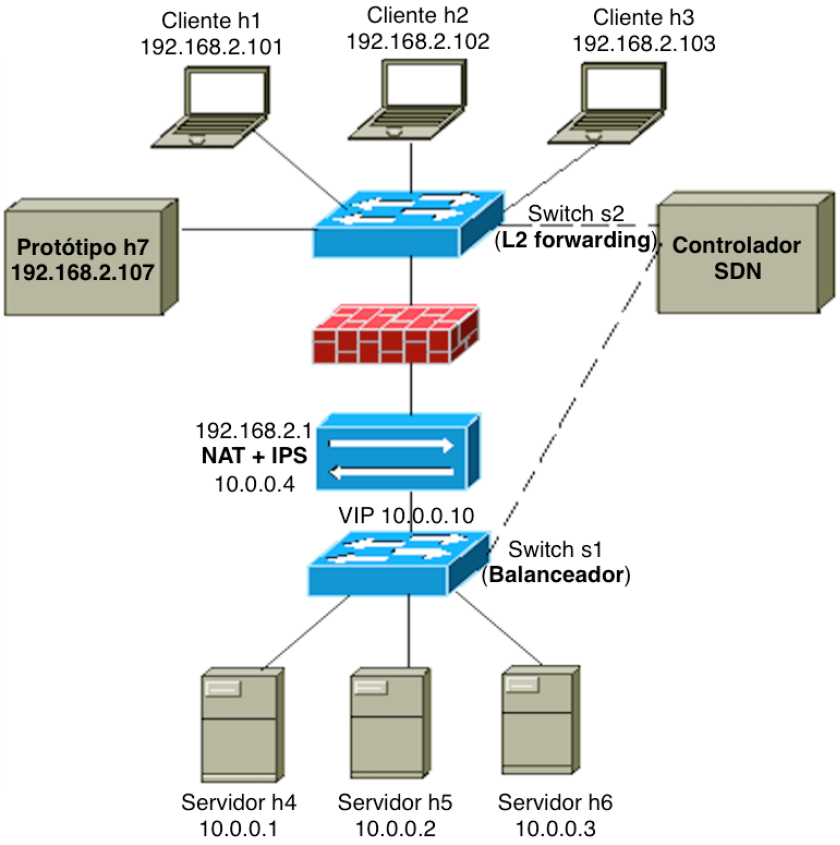



Figura 5.3: Topologia utilizada nos testes finais do protótipo.

há apenas um controlador POX atuando na rede. Na última avaliação, são feitos testes para as duas aplicações (Web e compartilhamento de arquivos) simultaneamente, com dois controladores POX atuando na rede. O primeiro controlador é executado com a porta padrão do OpenFlow para comunicação com os dispositivos de rede (6633), e aplica as regras do balanceador de carga e do *firewall* sobre o *switch* s1. O segundo controlador é executado na porta 6634, e aplica as regras de encaminhamento de pacotes sobre o *switch* s2. Os servidores h4, h5 e h6 atuam como servidores de arquivos (respondendo na porta 12345) e também como servidores Web CGIHTTPServer (respondendo na porta 8000).

As métricas de validação dos resultados do serviço Web são as mesmas dos testes anteriores. Para validação dos resultados com o compartilhamento de arquivos, as métricas obtidas pelo protótipo são comparadas aos seguintes **dados de validação**:

- O próprio *script* de testes para o controle da disponibilidade e dos parâmetros dos middleboxes;
- Capturas nas interfaces do ambiente emulado no Mininet utilizando o Wireshark para validar os dados de conexão obtidos pelas análises de capturas do controlador SDN;
- Os resultados armazenados em um arquivo texto pelo *script* configurado nos *hosts* clientes, para validação dos tempos de resposta obtidos.

Da mesma forma que nas avaliações parciais, para as avaliações seguintes também foram definidos valores de referência para as diferenças entre os tempos de resposta do

protótipo e dos dados de validação. Os resultados de tempo de resposta do protótipo serão considerados válidos caso as diferenças entre os tempos sejam menores do que um **valor de referência** computado como 10% do tempo de resposta médio de validação.

### 5.3.1 Teste com Aplicação Web

Neste teste, o protótipo monitora as informações das conexões realizadas dos *hosts* h1, h2 e h3 para os servidores Web h4, h5 e h6, que respondem por meio de um endereço IP virtual (do Inglês, *Virtual IP Address*, VIP) do balanceador de carga. O *firewall* realiza o bloqueio de todas as conexões entre o endereço IP do *host* h1 e o VIP do balanceador. Assim, suas conexões com o serviço Web devem constar como não completadas nos testes. O IPS, por sua vez, bloqueia todas as conexões do *host* h3 na porta 8000. Dessa forma, as conexões de h3 com o serviço Web também devem constar como não completadas. Já as conexões do host h2 não possuem nenhum tipo de bloqueio, devendo ser bem-sucedidas.

Todas as conexões a partir dos *hosts* h1, h2 e h3 para o serviço Web são realizadas pelo HTTPing, que faz sempre requisições de um arquivo de 3900 *bytes*. O *host* h1 faz 20 requisições, o h2 faz 1000 requisições e o h3 faz 100 requisições. Para cada um dos *hosts*, as requisições são espaçadas de 1 segundo. Os três *hosts* clientes iniciam seus conjuntos de requisições praticamente ao mesmo tempo. Espera-se que o protótipo interprete as conexões de h1 e h3 como bloqueadas, e que obtenha tempos de resposta para as conexões de h2 compatíveis com os resultados do HTTPing. A disponibilidade da aplicação será definida pela proporção de conexões completadas, que deve ser igual a $(1000/1120) = 89,29\%$, segundo o *script* deste teste. As métricas de rede das conexões são validadas por uma captura em todas as interfaces do Mininet realizada no Wireshark, de forma independente das capturas do protótipo.

Tabela 5.9: Principais resultados do teste final com a aplicação Web.

| | | Host 1 | Host 2 | Host 3 | Total |
|---|---|---|---|---|---|
| Protótipo | Tempo Resposta (ms) | – | 52,43 | – | 52,43 |
| | Disponibilidade (%) | 0 | 100 | 0 | 89,29 |
| Dados de Validação | Tempo Resposta (ms) | – | 52,12 | – | 52,12 |
| | Disponibilidade (%) | 0 | 100 | 0 | 89,29 |
| Diferença T. Resp. | Média Absoluta (ms) | – | 0,65 | – | 0,65 |
| | Desvio Padrão (ms) | – | 3,31 | – | 3,31 |
| | Valor de Referência (ms) | – | 5,21 | – | 5,21 |

Os resultados dessa avaliação, exibidos na Tabela 5.9, foram semelhantes aos dos testes anteriores. A disponibilidade calculada pelo protótipo foi idêntica à planejada pelo *script*, resultando em um valor de 89,29%, visto que as 20 conexões de h1 e as 100 conexões de h3 não foram completadas devido a bloqueios pelo *firewall* e pelo IPS, respectivamente. Entre os tempos de resposta do protótipo e do HTTPing houve uma diferença um pouco maior do que nos testes anteriores. Considerando-se apenas os tempos de resposta de h2 (já que as demais conexões não são completadas), a diferença média absoluta entre os tempos de resposta em milissegundos foi de 0,65, e o desvio padrão das diferenças foi de 3,31. Esses valores representam uma variação média absoluta de 1,24% no tempo de

resposta, e um desvio padrão de 6,31%. Ainda que esses valores sejam maiores do que os dos testes anteriores, ambos estão abaixo do valor de referência (10%), e portanto os tempos de resposta obtidos são válidos.

Observando-se as conexões individuais registradas na base de dados, notou-se que 2 das 1000 conexões de h2 para a VIP do balanceador foram capturadas de forma incompleta pelo *dumpcap* do protótipo. Ou seja, o que ocorreu para essas duas conexões foi que a captura que estava em andamento no momento de seus inícios foi interrompida antes de seu término. Esse tipo de erro poderia ter sido evitado caso se configurasse um intervalo maior para as capturas, penalizando porém o protótipo em seu intervalo de disponibilização dos dados. Para essas duas conexões, a duração (de onde se calcula o tempo de resposta) e a quantidade de *bytes* trafegados foram considerados menores do que deveriam, mas todos os demais parâmetros foram compreendidos corretamente. Excetuando-se essas duas conexões, a diferença média absoluta entre os tempos de resposta cairia para 0,51 (0,97%), e o desvio padrão das diferenças para 1,19 (2,27%). Em todo caso, são diferenças muito pequenas, e inferiores aos valores de referência com que determinamos se as informações de tempo de resposta são consistentes.

Os demais dados obtidos pelo protótipo foram iguais aos de validação (IPs e portas de origem/destino, quantidade de *bytes* trocados[1], *status* do *pool* de servidores, servidor que atendeu à conexão, *status* de bloqueio do IPS, *status* de bloqueio do *Firewall*).

### 5.3.2 Teste com Compartilhamento de Arquivos

Neste teste, foi configurado um serviço de compartilhamento de arquivos nos *hosts* h4, h5 e h6, desenvolvido em Python. Um arquivo de 5 MB é compartilhado com os *hosts* clientes na porta 12345 (no caso, os *hosts* h1, h2 e h3). Os *hosts* clientes fazem requisições periódicas desse arquivo, sem que exista nenhum mecanismo de *cache* (todos os *bytes* são enviados novamente a cada requisição). O protótipo monitora as informações das conexões entre os *hosts* clientes e os servidores de arquivos h4, h5 e h6, que respondem por meio de um VIP do balanceador de carga. Configura-se o *firewall* para bloquear todas as conexões de/para h3, e o IPS para bloquear as conexões de h2 para a porta do compartilhamento (12345). Assim, apenas as conexões do *host* h1 devem ser bem-sucedidas, e as demais devem ser bloqueadas.

As requisições do arquivo pelos *hosts* h1, h2 e h3 são realizadas por um *script* a cada 5 segundos, e o *script* mede o tempo desde o início da requisição até o recebimento completo do arquivo, armazenando esses registros de tempo em um arquivo texto. O *host* h1 faz 115 requisições, o h2 faz 10 requisições e o h3 faz 5 requisições. Os três *hosts* clientes iniciam seus conjuntos de requisições praticamente ao mesmo tempo. Espera-se que o protótipo interprete as conexões de h2 e h3 como bloqueadas, e que obtenha tempos de resposta para as conexões de h1 compatíveis com os registros de tempo armazenados no arquivo texto. A disponibilidade da aplicação será definida pela proporção de conexões completadas, que deve ser igual a $(115/130) = 88,46\%$, segundo o *script* deste teste. As métricas de rede das conexões são validadas por uma captura em todas as interfaces do Mininet realizada no Wireshark, de forma independente das capturas do protótipo.

[1]Excetuando-se as duas conexões interrompidas, que apresentaram uma quantidade de *bytes* inferior ao esperado.

Tabela 5.10: Principais resultados do teste final com o compartilhamento de arquivos.

| | | Host 1 | Host 2 | Host 3 | Total |
|---|---|---|---|---|---|
| Protótipo | Tempo Resposta (ms) | 413,67 | – | – | 413,67 |
| | Disponibilidade (%) | 100 | 0 | 0 | 88,46 |
| Dados de Validação | Tempo Resposta (ms) | 413,12 | – | – | 413,12 |
| | Disponibilidade (%) | 100 | 0 | 0 | 88,46 |
| Diferença T. Resp. | Média Absoluta (ms) | 1,63 | – | – | 1,63 |
| | Desvio Padrão (ms) | 4,58 | – | – | 4,58 |
| | Valor de Referência (ms) | 41,31 | – | – | 41,31 |

Os resultados dessa avaliação, exibidos na Tabela 5.10, foram bastante semelhantes aos iniciais. A disponibilidade calculada pelo protótipo foi idêntica à planejada pelo *script*, resultando em um valor de 88,46%, visto que as 10 conexões de h2 e as 5 conexões de h3 não foram completadas devido a bloqueios pelo *firewall* e pelo IPS. Entre os tempos de resposta medidos pelo *script* que era executado nos clientes e pelo protótipo, a diferença também foi muito pequena. Considerando-se apenas as conexões completadas (todas as de h1), a diferença média absoluta entre os tempos de resposta em milissegundos foi de 1,63, e o desvio padrão das diferenças foi de 4,58. Esses valores representam uma variação média absoluta de 0,39% no tempo de resposta, e um desvio padrão de 1,11%. Essas diferenças são muito pequenas e inferiores ao valor de referência, podendo-se considerar as informações de tempo de resposta do protótipo também consistentes. Para os demais parâmetros de conexão, validados a partir de comparação com os dados de uma captura independente no Wireshark, todos os resultados foram idênticos.

### 5.3.3 Teste com Múltiplos Controladores SDN e Aplicações

Este teste avalia de forma mais completa o protótipo, pois mostra sua capacidade de atuação com múltiplas aplicações (Web e compartilhamento de arquivos) e com mais de um controlador SDN (dois controladores POX). Nele, o protótipo monitora as informações das conexões realizadas dos *hosts* clientes h1, h2 e h3 para os servidores h4, h5 e h6, que respondem por meio de um endereço VIP do balanceador de carga nas portas 8000 para o serviço Web e 12345 para o serviço de compartilhamento de arquivos. O *firewall* bloqueia todas as conexões entre o endereço IP do *host* h1 e o VIP do balanceador. Assim, suas conexões com os dois serviços devem constar como não completadas nos testes. O IPS, por sua vez, bloqueia todas as conexões do *host* h2 na porta 12345. Dessa forma, as conexões de h2 com o serviço de compartilhamento de arquivos também devem constar como não completadas. As conexões de h2 para o serviço Web e de h3 para os dois serviços não possuem nenhum tipo de bloqueio, devendo ser bem-sucedidas.

São executados *scripts* independentes para gerar as requisições da aplicação Web e da aplicação de compartilhamento, de forma que as duas aplicações são executadas ao mesmo tempo, mas têm dados de validação segregados. No contexto da aplicação Web, são feitas requisições de um arquivo de 5 KB a partir dos *hosts* h1, h2 e h3 para o endereço VIP, utilizando o HTTPing. O *host* h1 faz 5 requisições, e os *hosts* h2 e h3 fazem 200 requisições cada. Para cada um dos *hosts*, as requisições são espaçadas de 1 segundo. Os três *hosts*

clientes iniciam seus conjuntos de requisições praticamente ao mesmo tempo. Espera-se que o protótipo interprete as conexões de h1 como bloqueadas, e que obtenha tempos de resposta para as conexões de h2 e h3 compatíveis com os resultados do HTTPing. A disponibilidade da aplicação será definida pela proporção de conexões completadas, que deve ser igual a $(400/405) = 98,77\%$, segundo o *script* deste teste. As métricas de rede das conexões são validadas por uma captura em todas as interfaces do Mininet realizada no Wireshark, de forma independente das capturas do protótipo.

Para a avaliação com a aplicação de compartilhamento de arquivos, os *hosts* clientes fazem requisições periódicas de um arquivo de 100 KB para o endereço VIP. A cada requisição o arquivo completo é enviado novamente. Os *hosts* h1 e h2 fazem 5 requisições cada, e o *host* h3 faz 20 requisições. Espera-se que o protótipo interprete as conexões de h1 e h2 como bloqueadas, e as conexões de h3 como bem-sucedidas, com tempos de resposta compatíveis com os apresentados pelo programa de compartilhamento de arquivos. A disponibilidade interpretada pelo protótipo deverá ser igual a $(20/30) = 66,67\%$. Da mesma forma que para a aplicação Web, as métricas de rede são validadas com uma captura independente do Wireshark.

Tabela 5.11: Principais resultados do teste final com duas aplicações e dois controladores.

| | **Aplicação Web** | | | | |
|---|---|---|---|---|---|
| | | Host 1 | Host 2 | Host 3 | Total |
| Protótipo | Tempo Resposta (ms) | – | 54,68 | 56,14 | 55,41 |
| | Disponibilidade (%) | 0 | 100 | 100 | 98,77 |
| Dados de Validação | Tempo Resposta (ms) | – | 54,50 | 55,98 | 55,24 |
| | Disponibilidade (%) | 0 | 100 | 100 | 98,77 |
| Diferença T. Resp. | Média Absoluta (ms) | – | 0,24 | 0,24 | 0,24 |
| | Desvio Padrão (ms) | – | 0,30 | 0,28 | 0,29 |
| | Valor de Referência (ms) | – | 5,45 | 5,60 | 5,52 |
| | **Aplicação de Compart.** | | | | |
| | | Host 1 | Host 2 | Host 3 | Total |
| Protótipo | Tempo Resposta (ms) | – | – | 72,95 | 72,95 |
| | Disponibilidade (%) | 0 | 0 | 100 | 66,67 |
| Dados de Validação | Tempo Resposta (ms) | – | – | 72,20 | 72,20 |
| | Disponibilidade (%) | 0 | 0 | 100 | 66,67 |
| Diferença T. Resp. | Média Absoluta (ms) | – | – | 3,31 | 3,31 |
| | Desvio Padrão (ms) | – | – | 5,62 | 5,62 |
| | Valor de Referência (ms) | – | – | 7,22 | 7,22 |
| | **Diferença Total T. Resp.** | | | | |
| | Média Absoluta (ms) | **0,38** | | | |
| | Desvio Padrão (ms) | **1,39** | | | |
| | Valor de Referência (ms) | **5,60** | | | |

Os resultados dessa avaliação são exibidos na Tabela 5.11. Assim como em todos os testes anteriores, a disponibilidade identificada pelo protótipo foi a esperada, resultando nos valores de 98,77% e 66,67% para a aplicação Web e para a aplicação de comparti-

lhamento, respectivamente. Também de forma similar aos demais testes, foram notadas pequenas diferenças entre os tempos de resposta do protótipo e dos dados de validação. Considerando-se apenas as conexões completadas, para as conexões Web, o valor de diferença absoluta média foi de 0,24 ms (0,43% do tempo de resposta médio), e o desvio padrão das diferenças foi de 0,29 ms (0,52%). Esses valores são muito pouco significativos, ou seja, os dados do protótipo foram praticamente iguais aos de validação.

Para as conexões do compartilhamento de arquivos, esses valores foram um pouco maiores: média absoluta de 3,31 ms (4,5%) e desvio padrão de 5,62 ms (7,7%). Notou-se que uma das conexões de h3 sofreu uma alteração na estatística de duração, por estar em execução no intervalo entre a realização de uma captura e a seguinte, provocando esse aumento na diferença do tempo de resposta. Como as conexões de compartilhamento de arquivos são mais longas, naturalmente há um maior risco de suas estatísticas sofrerem alterações pela mudança de arquivos de captura no protótipo. Excluindo-se essa conexão da lista, haveria uma redução significativa dessa diferença do tempo de resposta. De qualquer forma, mesmo considerando essa conexão, os valores das médias e desvios das diferenças entre os tempos de respostas são inferiores aos valores de referência.

No total de conexões de ambas aplicações houve uma diferença absoluta média de 0,38 ms, com desvio padrão de 1,39 ms, ambos inferiores ao valor de referência (5,60 ms). Assim, pode-se considerar as informações de tempo de resposta do protótipo também consistentes. Os demais dados obtidos pelo protótipo foram sempre idênticos aos de validação (IPs e portas de origem/destino, quantidade de *bytes* trocados[2], *status* do *pool* de servidores, servidor que atendeu à conexão, *status* de bloqueio do IPS, *status* de bloqueio do *firewall*). Assim, este teste demonstrou a capacidade do protótipo de trazer informações de gerenciamento consistentes na presença de múltiplos controladores SDN e aplicações.

[2]Excetuando-se a conexão de compartilhamento de h3 que foi interrompida.

# Capítulo 6

# Conclusão

Este trabalho propôs uma nova arquitetura de monitoramento para SDN, com foco em atacar os desafios trazidos pela presença ampla e diversa de middleboxes nas redes corporativas. Sugeriu-se que os middleboxes, por atuarem sobre o tráfego de forma variada e muitas vezes imprevisível, deveriam também ser monitorados por ferramentas de gerenciamento de desempenho de aplicações. A visibilidade do funcionamento dos middleboxes se daria a partir de um conjunto de métricas padronizado por tipo/funcionalidade e obtido por meio de APIs, em consonância com a tendência de programabilidade de dispositivos em SDN. Em adição às métricas de middleboxes, sugeriu-se a coleta de fluxos de rede a partir dos controladores de rede SDN. Como os controladores têm domínio total sobre os fluxos da rede, o trabalho de coleta de fluxos a partir desses pontos se torna extremamente simplificado em relação ao monitoramento tradicional, em que normalmente é necessário "instrumentar" diversos segmentos e dispositivos da rede.

A partir da modelagem conceitual da arquitetura de monitoramento, foi desenvolvido um protótipo de gerenciamento de desempenho. Esse protótipo consolida as informações dos middleboxes e dos dados de fluxos dos controladores de rede SDN em uma base de dados de forma dinâmica. Ele é capaz de avaliar a disponibilidade e o tempo de resposta de um conjunto de aplicações, e de exibir para cada conexão de uma aplicação as informações de estado dos middleboxes pelas quais ela trafega, bem como algumas informações de rede da conexão, como os endereços IP de origem e destino, as portas de origem e destino, a quantidade de dados trafegados e a sua duração. Para a disponibilização dos dados do protótipo ao administrador da solução, foi desenvolvida uma interface Web que exibe de forma amigável as principais informações de desempenho das aplicações e das suas conexões.

Na implementação do protótipo, foram desenvolvidas APIs para três middleboxes: um *firewall*, um IPS e um balanceador de carga. Essas APIs possibilitam o envio das informações de estados desses middleboxes para o protótipo. Com elas, o protótipo é capaz de interpretar informações desses três tipos de middleboxes concorrentemente, além de poder receber dados de fluxos de redes de múltiplos controladores, e tratar/exibir informações de múltiplas aplicações. O protótipo foi validado em três diferentes topologias no ambiente de emulação Mininet, em um total de nove avaliações. Essas avaliações englobaram a presença desses três middleboxes e de um NAT, e incluíram um teste com múltiplos controladores e aplicações. Em todos os testes, o protótipo trouxe resultados bastante consistentes, e praticamente não onerou o ambiente de emulação. Assim, provou-

se que ele é uma ferramenta de gerenciamento de desempenho confiável e que pode ser aplicada a diversos ambientes.

Ainda que o monitoramento implementado no protótipo seja simplificado, o conceito da arquitetura proposta pode ser estendido a sistemas mais complexos, que incluam informações de mais fontes, como dados específicos aos servidores de aplicação, ou informações obtidas a partir de *traps* SNMP, por exemplo. O protótipo demonstrado não se propõe a substituir ferramentas de monitoramento tradicionais, mas sim a simplificar a forma como algumas das informações necessárias a essas ferramentas são obtidas, e a trazer maior visibilidade dos middleboxes de uma rede, que tradicionalmente não são foco de ferramentas de gerenciamento de desempenho, apesar de sua grande relevância. De fato, o protótipo mostrou extrair informações extremamente úteis de aplicações e de conexões levando em consideração a influência dos middleboxes, a partir de modificações mínimas no ambiente monitorado. Os bons resultados obtidos com o protótipo mostram que a arquitetura proposta é uma forma de monitoramento inovadora e viável para SDN, especialmente no que diz respeito a contornar as dificuldades trazidas pela presença de middleboxes.

Em trabalhos futuros, espera-se validar o protótipo desenvolvido em redes reais de maior porte. Para tanto, propõe-se adaptar o protótipo de gerenciamento de desempenho para um formato mais escalável, em que se distribuiriam as tarefas de captura de tráfego, tratamento de capturas e tratamento de dados de middleboxes entre um conjunto de servidores. Espera-se, ainda, expandir o modelo de arquitetura proposto para abranger um conjunto maior de middleboxes e de métricas. Pretende-se expandir o conjunto de métricas de rede a partir de um tratamento mais aprofundado das capturas de tráfego de rede dos controladores, e também expandir o conjunto de métricas de aplicação a partir da integração com soluções de APM já existentes. Essa última expansão teria foco na obtenção de dados que requerem a "instrumentação" de servidores de aplicação (por exemplo, detalhes do funcionamento do *back-end* de uma aplicação, como o desempenho de porções do código em um servidor). Tais contribuições permitiriam que o protótipo se tornasse uma ferramenta ainda mais útil e verdadeiramente competitiva no mercado de gerenciamento de desempenho para SDN.

# Referências


[1] Pyro 4. Pyro 4.38 documentation. https://pythonhosted.org/Pyro4/intro.html, Acessado em 01/07/2015. 39

[2] SteelCentral Packet Analyzer. Graphical console for high-speed packet analysis. http://www.riverbed.com/products/performance-management-control/network-performance-management/High-Speed-Packet-Analysis.html, Acessado em 08/02/2014. 20

[3] J. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat. xOMB: Extensible Open Middleboxes With Commodity Servers. In *ANCS*, 2012. 15

[4] Apache. HTTP Server Project. http://httpd.apache.org, Acessado em 10/07/2015. 36

[5] OpenFlow Archive. Learn more. http://archive.openflow.org/wp/learnmore/, Acessado em 04/05/2014. 3, 9

[6] OpenFlow Archive. Tutorial Openflow. http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial, Acessado em 04/05/2014. ix, 12

[7] Bootstrap. Templates. http://www.prepbootstrap.com/, Acessado em 06/07/2015. 36

[8] Brocade. Virtual Traffic Manager. http://www.brocade.com/en/products-services/application-delivery-controllers.html, Acessado em 06/08/2015. 15

[9] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues. *RFC 3234*, 2002. 2, 13

[10] SDX Central. SDN and NFV APIs and SDKs. https://www.sdxcentral.com/comprehensive-list-of-sdn-apis/, Acessado em 09/01/2015. x, 23

[11] SDX Central. What are SDN Southbound APIs? https://www.sdxcentral.com/resources/sdn/southbound-interface-api/, Acessado em 30/04/2015. 8

[12] SDX Central. What are SDN Northbound APIs? https://www.sdxcentral.com/resources/sdn/north-bound-interfaces-api/, Acessado em 30/06/2015. 8

[13] CGIHTTPServer. CGI-capable HTTP request handler. https://docs.python.org/2/library/cgihttpserver.html, Acessado em 20/03/2015. 40

[14] Citrix. NetScaler. http://www.citrix.com.br/products/netscaler-application-delivery-controller/, Acessado em 06/05/2014. 15

[15] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. Ettm: A scalable fault tolerant network manager. In *NSDI*, 2011. 15

[16] Dumpcap. Dump network traffic. https://www.wireshark.org/docs/man-pages/dumpcap.html, Acessado em 15/03/2015. 33

[17] J. Edelman. Are there alternatives to the OpenFlow protocol? http://searchsdn.techtarget.com/answer/Are-there-alternatives-to-the-OpenFlow-protocol, Acessado em 03/04/2014. 8

[18] F5. BIG-IP. https://f5.com/products/big-ip, Acessado em 06/05/2014. 15

[19] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The Road to SDN. *Queue*, 11(12):20:20–20:40, 2013. 3

[20] POX L2 Firewall. Firewall for SDN. https://gist.github.com/wh5a/6015943, Acessado em 10/05/2015. 28

[21] Flask. API. http://flask.pocoo.org/docs/0.10/api/, Acessado em 05/07/2015. 33

[22] Karl Flinders. Monitoring not enough for firms where IT and business are indistinguishable. http://www.computerweekly.com/news/2240209038/Monitoring-not-enough-for-firms-where-IT-and-business-are-\indistinguishable, Acessado em 01/07/2015. 1

[23] Project Floodlight. Floodlight Framework. http://www.projectfloodlight.org, Acessado em 25/06/2015. 11

[24] Gartner. Application Performance Monitoring. http://blogs.gartner.com/jonah-kowall/2014/05/29/criteria-for-the-2014-magic-quadrant-for-\application-performance-monitoring/, Acessado em 20/08/2014. 14

[25] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella. Toward software-defined middlebox networking. In *Hotnets*, 2012. 3, 5, 15, 16, 21

[26] J. Giacomoni. Extending SDN architectures with F5's L4-7 Application and Gateway Services. Whitepaper, F5, 2015. x, 23

[27] Google. Grupo de pesquisa em redes. http://research.google.com/pubs/Networking.html, Acessado em 25/08/2014. 3

[28] Carlos A. Heuser. *Projeto de Banco de Dados*. Editora Sagra, 4 edition, 1998. ix, 24, 25

[29] HTTPing. Linux man page. http://linux.die.net/man/1/httping, 2015. Acessado em 21/02/2015. 40

[30] JSON. Introducing json. http://www.json.org, Acessado em 08/07/2015. 22

[31] D. Mattos. Xenflow: Um sistema de processamento de fluxos robusto e eficiente para redes virtuais. Technical report, Technical report, Deparamento de Eletrônica e de Computação. UFRJ - Universidade Federal do Rio de Janeiro, 2011. viii, 17, 19, 43. 7

[32] Mininet. An Instant Virtual Network on your Laptop. http://mininet.org, Acessado em 01/07/2014. 11, 12, 27

[33] T. Nadeau, D. Thomas, and K. Gray. *SDN: Software Defined Networks*. O'Reilly, Sebastopol, 2013. ix, 7, 9, 10, 11

[34] Open Networking. Open Networking Foundation. https://www.opennetworking.org/, Acessado em 20/04/2014. 8

[35] C. Paasch and O. Bonaventure. Multipath tcp. *Communications of the ACM*, 57(4):51–57, Abril 2014. ix, x, 2, 22

[36] M. Panwar. Application performance management emerging trends. *International Conference on Cloud and Ubiquitous Computing and Emerging Technologies*, 2013. 1

[37] pgAdmin. PostgreSQL Tools. http://www.pgadmin.org, Acessado em 01/07/2015. 31

[38] PostgreSQL. PostgreSQL Database. http://www.postgresql.org, Acessado em 20/06/2015. 29

[39] Python Pypcap. Pypcap 1.1.3 - Packet capture library. https://pypi.python.org/pypi/pypcap, Acessado em 06/07/2015. 39

[40] Z. Qaziy, C. Tuy, R. Miao, L. Chiangy, V. Sekary, and M. Yu. Practical and incremental convergence between sdn and middleboxes. In *Open Networking Summit*, 2013. 16

[41] Radware. Fastview. http://www.radware.com/Products/FastView/, Acessado em 06/05/2014. 15

[42] NOX Repository. NOX/POX Website. http://www.noxrepo.org/, Acessado em 31/03/2014. 10, 11, 27, 28

[43] Riverbed. Riverbed SteelCentral. http://www.riverbed.com/products/performance-management-control/, Acessado em 12/08/2014. 16

[44] C. Rothenberg, M. Nascimento, M. Salvador, and M. Magalhães. Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. *Cad. CPqD Tecnologia, v. 7, n.1, p.*, 7(1):65–76, Jun. 2011. 3, 7

[45] Ryu. Component-based Software Defined Networking Framework. http://osrg.github.io/ryu/, Acessado em 26/06/2015. 11

[46] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *NSDI*, 2012. 15

[47] Z. Shang, W. Chen, Q. Ma, and B. Wu. Design and implementation of server cluster dynamic load balancing based on openflow. In *International Joint Conference on Awareness Science and Technology and Ubi-Media Computing (iCAST-UMEDIA)*. IEEE, 2013. 15

[48] Snort. Intrusion Detection/Prevention Software. `https://www.snort.org`, Acessado em 03/02/2015. 28

[49] Stanford. Grupo de pesquisa em SDN. `http://onrc.stanford.edu`, Acessado em 25/08/2014. 3

[50] M. Stiemerling, J. Quittek, and C. Cadar. IETF RFC 4540. `http://tools.ietf.org/html/rfc4540`, Acessado em 10/08/2014. 15

[51] Visual TruView. Monitoração Unificada de Rede e do Desempenho de Aplicativos. `http://pt.flukenetworks.com/apps/truview`, Acessado em 08/07/2015. 20

[52] TShark. Dump and analyze network traffic. `https://www.wireshark.org/docs/man-pages/tshark.html`, Acessado em 03/07/2015. 20, 35

[53] Hardeep Uppal and Dane Brandon. Openflow based load balancing. *CSE561: Networking - Project Report*, 2010. 15

[54] VMware. Vmware nsx. `http://www.vmware.com/products/nsx`, Acessado em 20/07/2014. 3

[55] Richard Wang, Dana Butnariu, and Jennifer Rexford. Openflow-based server load balancing gone wild. *Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2011. 15

[56] Alan Weissberger. Open Network Foundation and Other Organizations; ONF-Optical Transport WG; Ciena and SDN. `http://viodi.com/2013/04/24/open-network-foundation-onf-optical-transport-wg-ciena-sdn/`, Acessado em 01/07/2015. ix, 4

[57] PostgreSQL Wiki. Psycopg2 Tutorial. `https://wiki.postgresql.org/wiki/Psycopg2_Tutorial`, Acessado em 05/07/2015. 33

[58] WSGI. Web Server Gateway Interface. `http://wsgi.readthedocs.org/en/latest/learn.html`, Acessado em 05/07/2015. 33

# Lista de Abreviaturas e Siglas

**API** Application Programming Interface.

**APM** Application Performance Management.

**BGP** Border Gateway Protocol.

**CGI** Common Gateway Interface.

**CSS** Cascading Style Sheets.

**DNS** Domain Name System.

**HPC** High Performance Computing.

**HTML** Hypertext Markup Language.

**HTTP** Hypertext Transfer Protocol.

**IDS** Intrusion Prevention System.

**IP** Internet Protocol.

**IPS** Intrusion Prevention System.

**JSON** JavaScript Object Notation.

**LLDP** Link Layer Discovery Protocol.

**MAC** Media Access Control.

**MPLS** Multiprotocol Label Switching.

**NAT** Network Address Translation.

**NAT-PT** Network Address Translation - Port Translation.

**NFS** Network File System.

**OSI** Open Systems Interconnection.

**OSPF** Open Shortest Path First.

**PAT** Port Address Translation.

**PHP** Hypertext Preprocessor.

**REST** Representational State Transfer.

**RFC** Request for Comments.

**RSPAN** Remote Switched Port Analyzer.

**SDK** Software Development Kit.

**SDN** Software-Defined Networking.

**SGBD** Sistema de Gerência de Banco de Dados.

**SNMP** Simple Network Management Protocol.

**SOCKS** Socket Secure.

**SPAN** Switch Port Analyzer.

**SQL** Structured Query Language.

**TAP** Test Access Point.

**TCP** Transmission Control Protocol.

**TI** Tecnologia da Informação.

**VIP** Virtual Internet Protocol address.

**VM** Virtual Machine.

**WAN** Wide Area Network.

**WMI** Windows Management Instrumentation.

**XML** Extensible Markup Language.

# Apêndice A

# Código Fonte do Protótipo

Os principais trechos do código fonte do protótipo, discutidos no Capítulo 4, são apresentados neste Apêndice.

## A.1 Criação de Tabela

Exemplo de instrução de criação de uma tabela de conexões em uma topologia com um *firewall*, um IPS e um balanceador de carga.

```
CREATE TABLE statisticsall
(
  "IPSrc" text NOT NULL,
  "IPDst" text NOT NULL,
  "PortSrc" integer NOT NULL,
  "PortDst" integer NOT NULL,
  "TotalBytes" integer NOT NULL,
  "Duration" numeric NOT NULL,
  "ConnStart" text NOT NULL,
  "ConnEnd" text NOT NULL,
  "StatusFW" boolean DEFAULT false,
  "StatusIPS" integer,
  "StatusLBPool" boolean,
  "ChosenServerLB" text,
  "LBPolicy" text,
  "Ready" boolean DEFAULT false,
  "AbsConnStart" numeric,
  "AbsConnEnd" numeric,
  CONSTRAINT "ConnIDAll" PRIMARY KEY ("ConnStart", "IPSrc", "
      IPDst", "PortSrc", "PortDst")
)
WITH (
  OIDS=FALSE
);
ALTER TABLE statisticsall
  OWNER TO userh7;
```

## A.2 Limpeza de Dados Antigos

Exemplo de instrução de deleção de linhas referentes a conexões antigas na tabela, de acordo com um período de retenção de três dias. A variável TempoAbsolutoAtual pode ser obtida pela função *time* em Python, e 259200 são a quantidade de segundos no período de três dias.

```
DELETE FROM statisticsall
 WHERE "AbsConnEnd" < TempoAbsolutoAtual - 259200;
```

## A.3 API do *Firewall*

Excerto de código da API em Python inserido em partes específicas do código do *firewall* para o envio de informações para o protótipo de gerenciamento de desempenho.

```
inport = event.port
packet = event.parsed
tcpp = packet.find('tcp')
if tcpp:
  ipp = packet.find('ipv4')
  if ipp:
    key = ipp.srcip,ipp.dstip,tcpp.srcport,tcpp.dstport
    status = "TRUE"
    url = "http://192.168.2.107:5000/alertfw"
    data = json.dumps({'src-ip':ipp.srcip,'src-port':tcpp.
       srcport,'dst-ip':ipp.dstip,'dst-port':tcpp.dstport,'
       time':str(time()),'status':status}, default=str)
    header = ["Content-Type: application/json"]
    te = pycurl.Curl()
    te.setopt(pycurl.URL, url)
    te.setopt(pycurl.POST, 1)
    te.setopt(pycurl.POSTFIELDS, data)
    te.setopt(pycurl.HTTPHEADER, header)
    te.perform()
```

Para a sua execução, é necessário importar algumas bibliotecas no código Python que executa o middlebox, conforme o código abaixo.

```
import pycurl, json
from time import time
```

## A.4 Recebimento das Informações de um Middlebox pelo Protótipo

Trecho de código em que o servidor de monitoramento recebe as informações do *firewall*.

```
@app.route('/alertfw', methods = ['POST'])
def alertfw():
  IPSrc = str(request.json['src-ip'])
  IPDst = str(request.json['dst-ip'])
  PortSrc = str(request.json['src-port'])
  PortDst = str(request.json['dst-port'])
  ConnTime = str(request.json['time'])
  Status = str(request.json['status'])
```

Para a sua execução, é necessário importar a biblioteca *json* e a biblioteca *request* do Flask.

## A.5 Servidor Web do Protótipo

Código que executa o servidor Web do protótipo de gerenciamento de desempenho. O objeto definido como *app* abaixo é um servidor Web implementado a partir do Flask.

```
scheduler = scheduler(time, sleep)
app = Flask(__name__)
app.run(host=<endereco-monitoramento>)
```

Para a execução do código acima, são importadas algumas bibliotecas, da seguinte forma:

```
from flask import Flask
from flask import request
```

## A.6 Armazenamento de UPDATES do *Firewall*

Fragmento do código da função *alertfw*, executada no protótipo para tratar as informações recebidas do *firewall*. A instrução de atualização da base de dados com as informações recebidas do *firewall* é armazenada em um arquivo, que será lido e "reciclado" de tempos em tempos.

```
@app.route('/alertfw', methods = ['POST'])
def alertfw():
  # more code above #
  abstimenow = float(ConnTime)
  query = "UPDATE statisticsall SET \"StatusFW\" = %s, \"Ready\"
     = TRUE WHERE (\"IPDst\" = '%s') and (\"PortSrc\" = '%s') 
     and (\"PortDst\" = '%s') and (%f > \"AbsConnStart\" - 60) 
     and (%f < \"AbsConnEnd\" + 60);" % (Status,IPDst,PortSrc,
     PortDst,abstimenow,abstimenow)
  mbfile = open("/home/snort/scripts/testes/stats_mb", "a")
  mbfile.write(query)
  # more code below #
  return "Alert Firewall Ok"
```

## A.7 Conexão e Atualização da Base de Dados

Funções de conexão com a base de dados e de atualização dessa base a partir do arquivo de instruções UPDATE dos middleboxes. É necessária a importação da biblioteca Psycopg2 para a sua execução.

```
def process_mb_file(mbfile):
  latest_conn_time = ''
  try:
    mbf = open(mbfile)
  except IOError:
    print "Unable to open file " + mbfile
  else:
    # Connect to the Database
    conn, cursor = connect_DB()
    # Get values from csv file and add them to the Database
    for i, line in enumerate(mbf):
      try:
        cursor.execute(line)
        print "Running query: " + line
      except psycopg2.IntegrityError:
        print "Integrity error!"
        conn.rollback()
      else:
        conn.commit()
    mbf.close()
    print "DB was updated with MB file"
    call("mv " + mbfile + " " + mbfile + ".read", shell=True)

def connect_DB():
  #Define our connection string
  conn_string = "host='192.168.2.254' dbname='dbh7' user='
     userh7' password='123'"

  print "Connecting to database\n     ->%s" % (conn_string)
  conn = psycopg2.connect(conn_string)
  cursor = conn.cursor()
  print "Connected!\n"

  return (conn, cursor)
```

## A.8 API do IPS Snort

Código em Python que executa a API do IPS Snort.

```
from idstools import unified2
import pycurl,json
```

```
from time import time, gmtime, strftime, localtime
from datetime import datetime
import sys, getopt

def check_args(argv):
  monitoring_host = ''
  try:
    opts, args = getopt.getopt(argv,"h:",["host="])
  except getopt.GetoptError:
    print 'Wrong usage! Run as:\nsnort_API.py -h <monitoring
       host ip>'
    sys.exit(2)
  for opt, arg in opts:
    if opt in ("-h", "--host"):
      monitoring_host = arg
    else:
      print 'Wrong usage! Run as:\nasnort_API.py -h <monitoring
         host ip>'
      sys.exit()
  if monitoring_host:
    print 'Monitoring host IP is ', monitoring_host
    return monitoring_host
  else:
    print 'Wrong usage! Run as:\nsnort_API -h <monitoring host
       ip>'
    sys.exit()


if __name__ == '__main__':

  global monitoring_host
  monitoring_host = check_args(sys.argv[1:])

  control = []
  url = "http://" + monitoring_host + ":5000/alertips"
  header = ["Content-Type: application/json"]
  c = pycurl.Curl()
  c.setopt(pycurl.URL, url)
  c.setopt(pycurl.POST, 1)
  c.setopt(pycurl.HTTPHEADER, header)

  reader = unified2.SpoolEventReader("/var/log/snort", "snort.
     alert", follow=True)

  for event in reader:
```

```
flow = str(event['source-ip'])+str(event['sport-itype'])+str
    (event['destination-ip'])+str(event['dport-icode'])
if flow in control:
  pass
else:
  control.append(flow)
  data = json.dumps({"src-ip":str(event['source-ip']),"src-
      port":str(event['sport-itype']),"dst-ip":str(event['
      destination-ip']),"dst-port":str(event['dport-icode']),
      "time":str(time()),"StatusIPS":str(event['blocked'])})
  c.setopt(pycurl.POSTFIELDS, data)
  c.perform()
```

## A.9 Cômputo de Estatísticas de Capturas

Trechos mais relevantes do código em Python que gera as estatísticas das capturas de rede. Ele também "recicla" o arquivo em que são armazenadas as atualizações provenientes dos middleboxes.

```
def run_tshark():
  try:
    oldest = min(glob.iglob('/home/snort/scripts/testes/*-
        pcapng'), key=os.path.getctime)
    print "Tshark will run on " + oldest
    if oldest != "":
      splittime = re.split("-", oldest)
      filetime = splittime[1]
      csvfile = "/home/snort/scripts/testes/stats_tshark-" +
          filetime + "-csv"
      call("tshark -r " + oldest + " -q -z conv,tcp -n | tail
        -n +6 | head -n -1 | tr ' ' ';' | sed 's/;*;/;/g' | 
        sed 's/<->;//g' | tr ':' ';' | sed 's/$/;/' | tr ',' 
        '.' >> " + csvfile, shell=True)
      # Process the file adding the new connections to the
          Database and mark the file as read
      process_stats_file(csvfile,filetime)
      call("mv " + oldest + " " + oldest + ".read", shell=True
          )

      try:
        mbfile = min(glob.iglob('/home/snort/scripts/testes/
            stats_mb*-old'), key=os.path.getctime)
        print "DB will be updated with " + mbfile
        if mbfile != "":
          Timer(float(capture_interval) + 20, process_mb_file,
              [mbfile]).start()
```

```
        except (ValueError, TypeError):
            print "No new stats_mb files to process"
    except (ValueError, TypeError):
        print "No new files for Tshark to process"
```

## A.10 Tratamento de Estatísticas de Capturas

Trechos mais relevantes do código em Python que trata os arquivos ".csv" que contêm as estatísticas das capturas de rede, para então inserir as informações resultantes na base de dados.

```
def process_stats_file(csvfile,filetime):
    csv = open(csvfile)
    # Connect to the Database
    conn, cursor = connect_DB()
    # Get values from csv file and add them to the Database
    for i, line in enumerate(csv):
        commvalues = re.split(";", line)
        IPSrc = commvalues[0]
        PortSrc = commvalues[1]
        IPDst = commvalues[2]
        PortDst = commvalues[3]
        TotalBytes = commvalues[9]
        RelStart = commvalues[10]
        Duration = commvalues[11]
        # Insert values into Database
        insert_DB(conn, cursor, IPSrc,PortSrc,IPDst,PortDst,
            TotalBytes,Duration,RelStart,filetime)
```

## A.11 Inserção na Base de Dados

Código em Python que insere as informações de uma conexão na base de dados.

```
def insert_DB(conn,cursor,IPSrc,PortSrc,IPDst,PortDst,
    TotalBytes,Duration,RelStart,filetime):
    # ATTENTION: query is ignoring uppercase for table name...
    if IPSrc and PortSrc and IPDst and PortDst and TotalBytes
        and Duration and RelStart and filetime:
        if float(Duration) != 0:
            absconnstart = float(filetime) + float(RelStart)
            connstart = datetime.fromtimestamp(absconnstart)
            absconnend = float(filetime) + float(RelStart) + float(
                Duration)
            connend = datetime.fromtimestamp(absconnend)
            query = "INSERT INTO statisticsall VALUES ('%s', '%s',
                '%s', '%s', '%s', '%s', '%s', '%s', FALSE, NULL, NULL
```

```
    , NULL, NULL, FALSE, '%s', '%s');" % (IPSrc, IPDst,
    PortSrc, PortDst, TotalBytes, Duration, connstart, connend
    , absconnstart, absconnend)
try:
  cursor.execute(query)
  print "Running query: " + query
except psycopg2.IntegrityError:
  print "Integrity error!"
  conn.rollback()
else:
  conn.commit()
```