UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Nilton Camargo Batista da Silva

SCPNET - UMA ARQUITETURA DE MONITORAMENTO DE REDE BASEADA EM POLÍTICAS

Santa Maria, RS
2018

Nilton Camargo Batista da Silva

SCPNET - UMA ARQUITETURA DE MONITORAMENTO DE REDE BASEADA EM POLÍTICAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Ciência da Computação**.

Orientador: Prof. Dr. Carlos Raniery Paula dos Santos

Santa Maria, RS

2018

Camargo Batista da Silva, Nilton

SCPnet - Uma arquitetura de monitoramento de rede baseada em políticas / por Nilton Camargo Batista da Silva. – 2018.
68 f.: il.; 30 cm.

Orientador: Carlos Raniery Paula dos Santos
Dissertação (Mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação , RS, 2018.

1. Monitoramento de redes. 2. Gerenciamento Baseado em política. 3. PBNM. 4. Big Data. I. Raniery Paula dos Santos, Carlos. II.SCPnet - Uma arquitetura de monitoramento de rede baseada em políticas.

E-mail: nbatista@inf.ufsm.br

Nilton Camargo Batista da Silva

SCPNET - UMA ARQUITETURA DE MONITORAMENTO DE REDE BASEADA EM POLÍTICAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do título de **Mestre em Ciência da Computação**.

Aprovado em 28 de fevereiro de 2018:

Carlos Raniery Paula dos Santos, Dr. (UFSM)
(Presidente/Orientador)

Raul Ceretta Nunes, Dr. (UFSM)

Jéferson Campos Nobre, Dr. (UNISINOS)

Santa Maria, RS

2018

DEDICATÓRIA

Dedico este trabalho a minha esposa Jéssica Baldissera Carollo,

aos meus pais Fábio e Raquel,

a minha irmã Brenda,

e também não podendo esquecer meu catioro Buzz,

que sempre acreditaram em mim e me apoiaram.

Amo vocês!

AGRADECIMENTOS

Primeiramente, agradeço à Deus, pois sem ele este trabalho não seria possível. Que com sua infinita sabedoria foi um importante guia na minha trajetória.

A minha amada esposa ***Jéssica****, por fazer parte da minha vida. Além de sempre estar ao meu lado, me dando apoio, carinho e me fazendo acreditar que consigo mais. Obrigado por sempre acreditar em mim.* ***Te amo pra sempre!***

Agradeço aos meus pais, ***Fábio e Raquel****, e a minha irmã* ***Brenda****, por todo amor e carinho que recebi durante a vida inteira e por sempre confiarem em mim.*

À toda minha família, por sua capacidade de acreditar e investir em mim.

Ao meu orientador ***Carlos Raniery Paula dos Santos****, pelo suporte, pelas correções, pelos incentivos e pelo acompanhamento durante o mestrado.*

Aos meus amigos da igreja do Bom Fim, que me incentivaram todos os dias e ofereceram apoio nos momentos críticos, em especial à meus amigos de coração ***José Carlos Feldmann do Amaral e Giuliano Iop Laporta*** *que sempre são amigos fiéis.*

A meus amigos do mestrado, pelos momentos divididos juntos, especialmente ao Anderson Monteiro, Vinícius Fulber, Thales Tavares e ao Leonardo Marcuzzo que se tornaram verdadeiras amigos e tornaram mais leve meu trabalho. Obrigada por dividir comigo as angústias e alegrias, e ouvirem minhas bobagens.

A todos aqueles que de alguma forma estiveram e estão próximos a mim, fazendo esta vida valer cada vez mais a pena.

“Alegrem-se na esperança,
sejam pacientes na tribulação e
perseverem na oração.”

(ROMANOS 12:12)

RESUMO

SCPNET - UMA ARQUITETURA DE MONITORAMENTO DE REDE BASEADA EM POLÍTICAS

AUTOR: NILTON CAMARGO BATISTA DA SILVA
ORIENTADOR: CARLOS RANIERY PAULA DOS SANTOS

As redes de computadores estão sempre permanente crescimento, o que causa um enorme aumento no número de dados e dificulta o gerenciamento. Nesse contexto, há um interesse recente em aplicar conceitos, ferramentas e tecnologias de *Big Data* em gerenciamento de redes. Juntamente com o grande volume de dados, um dos limitadores do gerenciamento de redes diz respeito à heterogeneidade de tecnologias utilizadas nas modernas redes de computadores. Deste modo, para que o administrador consiga uniformizar a gestão de redes é preciso que o sistema de gerenciamento utilizado seja capaz de interpretar os objetivos de forma abstrata. PBNM é uma técnica que utiliza o conceito de políticas para que o administrador de rede possa definir o que ele pretende realizar, utilizando para isso uma linguagem mais abstrata. Contudo, até agora, o uso de tais técnicas no contexto de monitoramento de redes ainda permanece desconhecido. Sendo assim, os principais objetivos desta dissertação é propor uma solução que permita ao administrador de redes expressar de forma abstrata os dispositivos e serviços que deseja monitorar em um ambiente capaz de lidar com grande número de dados. Para atingir tais objetivos nesta dissertação é proposto uma arquitetura de que permita ao administrador de redes expressar de forma abstrata os dispositivos e serviços que deseja monitorar em um ambiente capaz de lidar com grande número de dados. Com base nesta arquitetura foi desenvolvido um protótipo. Por fim, o protótipo desenvolvido, junto com o modelo de política para monitoramento de redes, foi avaliado em um ambiente de testes, onde foi possível perceber que ao usar políticas no monitoramento de redes existem algumas vantagens e que também não houve uma sobrecarga na rede de mensagens de requisição-resposta. Os resultados obtidos indicam que o uso de políticas juntamente com correlação de dados em um ambiente *Big Data* pode ser utilizado como mecanismo de monitoramento de redes.

Palavras-chave: Monitoramento de redes. Gerenciamento Baseado em política. PBNM. Big Data. Correlação de dados.

ABSTRACT

SCPNET - A POLICY-BASED NETWORK MONITORING ARCHITECTURE

AUTHOR: NILTON CAMARGO BATISTA DA SILVA
ADVISOR: CARLOS RANIERY PAULA DOS SANTOS

Computer networks are constantly growing, which causes a huge increase in the number of data and difficult their management. In this context, there is a recent interest in applying Big Data concepts, tools and technologies in network management. Along with the large volume of data, one of the limitations of network management concerns the heterogeneity of technologies used in modern computer networks. Therefore, for the administrator to achieve a uniform management of networks, it is necessary that the management system used would be able to interpret objectives in an abstract way. PBNM is a technique that uses the concept of policies so that the network administrator can define what are accomplishments to be achieved, using a more abstract language. Nevertheless, to date, the use of such techniques in the context of network monitoring remains unknown. Thus, the main objectives of this dissertation are to propose solutions which would allow the network administrator to express abstractly the devices and services that one wants to be monitored in an environment capable of handling large numbers of data. In order to achieve these objectives, an architecture is proposed in order to allow the network administrator to express, in an abstract way, the devices and services that are wanted to be monitored in an environment capable of handling large numbers of data. Based on this architecture, a prototype was developed. Finally, the prototype developed, along with the policy model for network monitoring, was evaluated in a test environment, where it was possible to perceive some advantages when using policies while monitoring networks, as there was there was no overload in the request-response messaging network. The results indicate that the use of policies together with data correlation in a Big Data environment can be used as a mechanism for monitoring networks.

Keywords: Network Monitoring. Policy-Based Management. PBNM. Big Data. Data Correlation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE APÊNDICES

LISTA DE ABREVIATURAS E SIGLAS

BI	*business intelligence*
CCA	*Candidate Corrective Action*
CEP	*Complex Event Processing*
COPS	*Common Open Police Service*
RSVP	*Resource Reservation Protocol*
DMTF	*Distributed Management Task Force*
DRM	*Dimension Reduction Module*
DSP	*Data Streaming Processor*
ECM	*Episode Classification Module*
EDM	*Episode Discovery Module*
EPL	*Eclipse Public License*
FTP	*File Transfer Protocol*
HTTP	*Hyper Text Transfer Protocol*
IoT	*Internet of Things*
IP	*Internet Protocol*
IETF	*Internet Engineering Task Force*
LDAP	*Lightweight Directory Access Protocol*
NEs	*Network Elements*
NOC	*Network Operations Center*
OSS	*Operations Support Systems*
PBNM	*Policy-Based Network Management*
PDP	*Policy Decision Point*
PEP	*Policy Enforcement Point*
PML	*Pattern Module Libery*
PMM	*Pattern Matching Module*
QoS	*Quality of Service*
RAP	*Resource Alocation Protocol*
RSM	*Recommender System Module*
SNMP	*Simple Network Management Protocol*
SQL	*Structured Query Language*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*

UDP	*User Datagram Protocol*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

As redes de computadores estão em permanente crescimento. Há a integração de inúmeras tecnologias (*e.g.*, *Internet of Things* - IoT) (MATOS; PAILLARD; CASTRO, 2016), e um progressivo aumento no número de usuários que começaram a utilizar as redes para distintas finalidades (MULAHUWAISH; BAKAR; GHAFOOR, 2012). Dessa forma, há uma crescente heterogeneidade e complexidade dos novos dispositivos adicionados as redes, um aumento no número de fluxos de dados gerados, com isso tornando a gerência de redes uma tarefa bastante complexa (LUCKHAM; SCHULTE, 2008).

Geralmente os administradores de redes se vêm obrigados a controlar uma combinação de equipamentos baseados em diferentes tecnologias, obrigando-os a controlar vários domínios de conhecimento diferentes com ferramentas de monitoramento tradicionais, o que normalmente gera uma sobrecarga na rede com mensagens de requisição-resposta. Assim, o interesse em uniformizar a gestão destes recursos e diminuir a sobrecarga na rede, é claramente importante. Esse cenário contribui para que a comunidade científica continue a desenvolver novas técnicas para prover um gerenciamento de redes eficaz (CASADO et al., 2012).

Em virtude da quantidade de dados a serem monitorados, alguns pesquisadores têm utilizados de técnicas de correlação de dados (*e.g., (SALAH; MACIÁ-FERNÁNDEZ; DÍAZ-VERDEJO, 2013)*), que consistem na interpretação investigativa de numerosos dados, conduzindo-os a um novo significado, com o intuito de reduzir a quantidade de dados transferidos aos administradores de rede.

Nesse contexto, ao mesmo tempo que a comunidade de gerência têm aplicado técnicas de correlação de dados no monitoramento de redes, há um interesse recente em aplicar conceitos, ferramentas e tecnologias de *Big Data* em gerenciamento de redes (*e.g.,* Os trabalhos de (CHENG et al., 2011), (ZAMAN et al., 2015), (ROBITZSCH et al., 2015)). As técnicas de *Big Data* estão sendo investigadas por inúmeros pesquisadores, tanto da academia como da indústria, o uso de tais técnicas apresentam desafios principalmente devido à natureza de seus dados: volumosos e evolutivos. Segundo GOPALKRISHNAN et al. (2012) uso de técnicas de *Big Data* dispõe de grande potencial para descoberta de novos conhecimentos e geração de *insigths*, com isso podendo guiar o gerenciamento de redes a um outro patamar, pois a análise pode ser realizada em tempo de execução, além de permitir uma análise inteligente.

Além do problema da quantidade de dados, um outro limitador para um processo de

gerência eficaz diz respeito à heterogeneidade de tecnologias utilizadas nas modernas redes de computadores. Dessa forma, para que o administrador de redes consiga otimizar seu trabalho, é necessário que o sistema de gerenciamento utilizado seja capaz de interpretar os objetivos a serem alcançados (BATISTA; FERNANDEZ, 2014). O gerenciamento de redes baseado em políticas (*Policy-based Network Management* - PBNM) utiliza o conceito de políticas para que o administrador de rede possa definir o que ele pretende realizar, utilizando para isso uma linguagem mais abstrata (SLOMAN, 1994). Além de, segundo BATISTA; FERNANDEZ (2014) o administrador de rede poder reutilizar uma mesma política. Segundo **?** PBNM tem demonstrado ser uma eficiente estratégia para simplificar a administração de sistemas complexos, normalmente caracterizados pela heterogeneidade e complexidade dos novos dispositivos. O objetivo das políticas é estabelecer um método de administração mais amigável à linguagem humana, promovendo o gerenciamento de todo o ambiente, ao invés de tratar da configuração individual de cada elemento da rede. O uso de políticas torna o sistema reutilizável e flexível. A modificação ou cancelamento de antigas políticas e a inclusão de novas políticas, permite que o sistema seja facilmente adaptado para atender a mudanças de requisitos.

No entanto, apesar das possíveis vantagens em utilizar PBNM no monitoramento de redes a fim de controlar uma combinação de equipamentos baseados em diferentes tecnologias e também não só usar PBNM com a finalidade de controlar e aplicar qualidade de serviço (*Quality of Service* - QoS) não há na literatura trabalhos que investiguem o uso dessa abordagem de forma adequada. Desta forma, o principal objetivo deste trabalho é propor uma solução que permita ao administrador de redes expressar de forma abstrata os dispositivos e serviços que deseja monitorar em um ambiente capaz de lidar com grande número de dados. Além disso, esse trabalho também conta com o desenvolvimento de algoritmos de correlação de dados, ações e de um modelo de política a ser usado no sistema.

O restante deste trabalho está organizado da seguinte forma. No Capítulo 2 é apresentado uma síntese sobre gerenciamento de redes baseado em políticas, monitoramento de redes e trabalhos relacionados, *Big Data* e também são apresentado os principais trabalhos relacionados. O capítulo 3 apresenta a arquitetura da solução proposta neste trabalho. Os detalhes da implementação da proposta são apresentados no Capítulo 4. No Capítulo 5 é apresentada a discussão da avaliação do uso da arquitetura em dois estudos de caso. Por fim, as considerações finais e os trabalhos futuros são apresentados no Capítulo 6.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os conceitos de PBNM e sua arquitetura de referência. Além disso, na subseção 2.2 são apresentados conceitos de monitoramento de redes e alguns trabalhos que utilizam *Policy-Based Network Management* (PBNM) no monitoramento de redes. A subseção 2.3 é realizado uma síntese sobre o tema *Big Data* e são apresentados trabalhos relacionados com monitoramento de redes em ambiente *Big Data*. Ao final do capítulo na seção 2.4 é realizado um breve resumo do mesmo.

2.1 GERENCIAMENTO DE REDES BASEADO EM POLÍTICAS - PBNM

O gerenciamento de redes convencional lida com um agrupamento de informação que cresce continuamente, tanto em heterogeneidade quanto em volume. Por essa razão tornando o gerenciamento de redes uma tarefa bastante complexa. Além de tudo, a circulação das informações é rigorosamente proporcional a dimensão da rede. A Internet possibilita a interconectividade dos usuários, com isso aumentando o número de informações a serem gerenciadas e também tornando as redes cada vez mais distribuída.

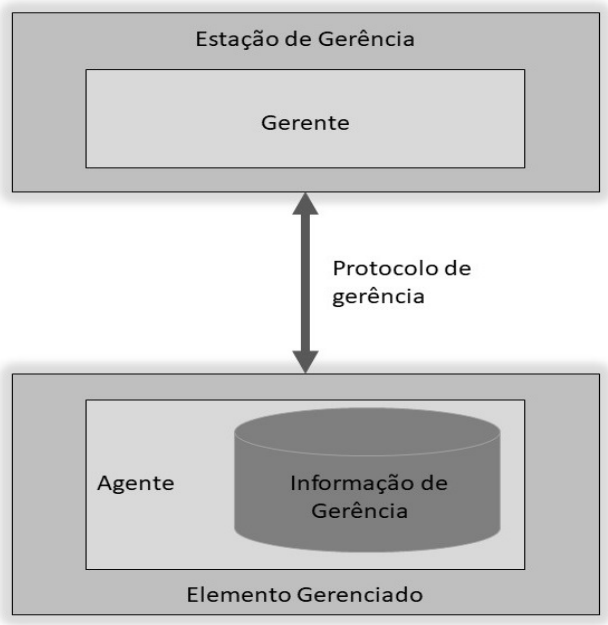
Segundo OLIVEIRA et al. (2017), o gerenciamento e o monitoramento de redes são tarefas extraordinariamente importantes para a vitalidade de uma rede de computadores, visto que, sem ações de gerenciamento, uma rede de computadores não tem como conservar-se operante por muito tempo. Além de atuarem reativamente, as tarefas de gerenciamento de redes também são pró-ativas no intuito de prevenir e identificar possíveis problemas.

De acordo com ZHOU et al. (2017), um sistema de gerenciamento de redes é constituída por $agentes$ realizando uma ação nos elementos gerenciados e por $gerentes$ realizando uma ação nas estações de gerenciamento. O termo $gerente$ pode ser empregado, também, para indicar a pessoa encarregada pelo gerenciamento da rede.

O gerenciamento de redes foi primeiramente impulsionada pela deficiência de monitoramento e controle do domínio de dispositivos que constituem as redes de comunicação. As primeiras abordagens de gerenciamento de rede foram baseadas no paradigma gerente-agente, como é demonstrado na Figura 1, onde o gerente centralizava as funções de monitoramento e controle, além de ser o encarregado pelo acesso aos inúmeros agentes (dispositivos) da rede. Já os agentes exerciam e ainda exercem o papel de abastecimento das variáveis do *Management*

Information Base (MIB) à medida que o gerente, por meio de métodos tipo *polling* controlam a rede. A finalidade deste método de gerenciamento era simplificar as tarefas do agente, possibilitando um desenvolvimento rápido e necessitando de poucos recursos dos equipamentos, tendo em vista que não era determinado nenhum mecanismo para a correspondência entre vários gerentes. Esta abordagem aumentava o volume de tráfego na rede, com isso degradando o tempo de resposta e também sobrecarregando o gerente com todas as funções.

Figura 1 – Arquitetura Básica de Gerência



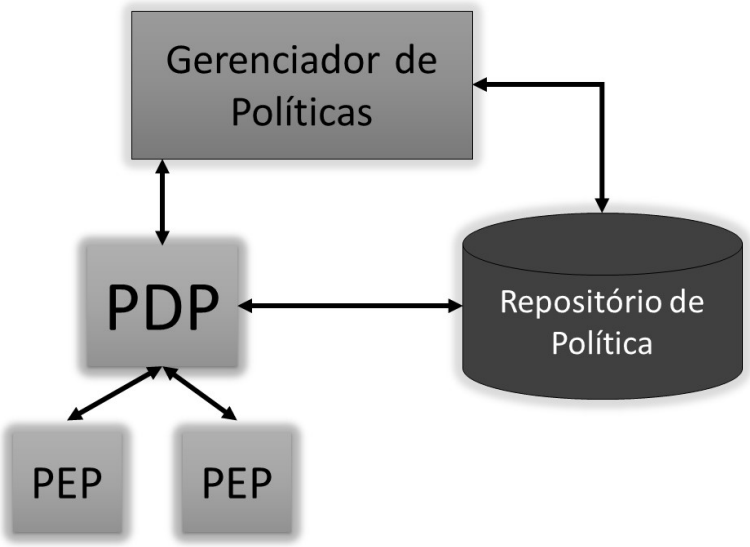
Fonte: acervo pessoal.

A gerência de rede convencional trabalha com uma coleção de informações de rede que cresce constantemente, tanto em heterogeneidade quanto em volume. Os administradores de redes investigam esse volume de dados recebidos pelas redes de computadores heterogêneas, primando sempre pela melhor performance de recursos e aplicações. Com isso uma solução que visa diminuir os problemas da gerência convencional de maneira parcialmente simples fundamentada em regras é o gerenciamento baseado em políticas. A gestão de redes baseado em políticas apresenta uma alternativa para superar muitas deficiências, contribuindo positivamente para as plataformas de gerenciamento convencionais. Seu uso tem como finalidade diminuir a complexidade, permitindo a automação de atividades na gerência de redes e a adaptabilidade às novas condições do ambiente.

2.1.1 Arquitetura de Referência

A arquitetura básica de PBNM foi descrita conforme *Internet Draft* [Stevens,1999a], como uma parcela do grupo de trabalho *framework* de políticas. A arquitetura básica é exposta na Figura 2, onde há a uma relação entre os diferentes dispositivos e políticas. Normalmente essa arquitetura é o padrão de sistemas PBNM mais aceita na maioria das literaturas (GRANVILLE et al., 2003).

Figura 2 – *Framework* de Políticas da IETF



Fonte: acervo pessoal.

O gerenciamento de rede baseado em políticas tem motivado amplas pesquisas nas últimas décadas dentro de um grupo de trabalho da IETF. O grupo de trabalho de *framework* de política da IETF criou o protocolo chamado *Common Open Police Service* (COPS), o qual originalmente é compatível com a integração de serviço. Atualmente é utilizada como o projeto geral de protocolo de políticas e tem diferentes extensões, assim como *Provisioning Extension* (COPS-PR) é utilizado para suportar configurações de gerenciamento. Os grupos da *Distributed Management Task Force* (DMTF) e IETF trabalham juntos para criar diferentes normalizações relacionadas a políticas: A IETF é encarregada pela arquitetura de definição e a DMTF define a normatização do modelo de informação. A *Microsoft e Cisco Systems* elaboraram uma infraestrutura comum de diretório capaz de unificar o gerenciamento dos produtos. Essa iniciativa analisa o esquema de diretório para integralizar a rede com serviços de diretórios e o modelo de

informação. Dessa forma ao invés de optar pelo gerenciamento de dispositivos individualmente, é factível definir regras de gerência para administrar seus recursos e a rede de forma centralizada. Os dois grupos de trabalho na IETF que estudam gerenciamento baseado em políticas são:

a) Grupo de trabalho do *framework* de políticas, que produz vários *Drafts* e *Request for Comments* (RFCs) para gerenciar, representar, dividir e reutilizar todas as políticas a caminho da interoperabilidade, escalabilidade e independência de *software* e *hardware* utilizada pela plataforma;

b) Grupo de trabalho de *Resource Alocation Protocol* (RAP), de quem preliminarmente foi o propósito de determinar a escalabilidade do modelo de gestão de política para *Resource Reservation Protocol* (RSVP). Este mesmo grupo estabeleceu um *framework* de políticas para o gestão de admissão, implementando uma interface de gerenciamento de políticas, os pontos de aplicações para configurar a política de decisão em elementos da rede e os pontos de decisão para distribuição de políticas, e o repositório para armazenamento.

A Figura 2 expressa os principais componentes do *framework* e os protocolos mais empregados para notificação de monitoramento e configuração de dispositivos.

a) Repositório de Políticas: É um depósito usado pelo *Policy Decision Point* (PDP) para recuperar políticas. As políticas são aprovisionadas em um alto nível, independentes dos elementos da rede e conforme o padrão de informação, o padrão indicado pela IETF é o *Commom Information Model*, mas repositórios como *Web servers*, diretórios ou banco de base de dados são também amplamente utilizados. O uso de um protocolo de acesso é solicitado com a necessidade de se incorporar aos outros componentes. A IETF aconselha o uso de *Lightweight Directory Access Protocol* (LDAP), mas outras soluções viáveis são o *Hyper Text Transfer Protocol* (HTTP), *Simple Network Management Protocol* (SNMP), *Secure Shell* (SSH) e outros.

b) Ferramenta de Gerenciamento de Políticas: Viabiliza a interface para fazer a gestão específica da rede e guardar as políticas em um repositório. Também atua de acordo com um monitor do estado, gerenciando redes por meio de políticas.

c) *Policy Decision Points* - São pontos no qual todas as decisões que devem ser aplicadas na rede são produzidas. Os PDPs verificam as políticas e informações sobre a condição

da rede para decidir quais políticas são fundamentais aplicar. Estas políticas são enviadas em conformidade com configuração dos dados para *Policy Enforcement Point* (PEP) ou PEPs correspondentes. A arquitetura considera a possibilidade de ter um componente PDP para um ou vários PEPs e estes com todas regras físicas dos dispositivos. O PDP é responsável por localizar o grupo de regras e aplicar a um PEP, recuperando do repositório e transformando-as dentro de um formato sintático que pode ser entendida por dispositivos e distribuído ao PEP. Neste sentido, o PDP atua conforme monitor do estado da rede, verificando se todas as solicitações de condição foram satisfeitas pela política de aplicação. Se o PDP justificar transformações, então deve carregar no repositório de políticas novamente.

d) *Policy Enforcement Points* - São elementos envolvidos na execução de políticas que o PDP ordena. Alguns exemplos desses PEPs são roteadores, *proxies*, *firewalls*, etc. Cada PEP contém a política na forma de configurações específicas de ações sendo responsável pela persistência no dispositivo. O PEP pode também comunicar ao PDP quando qualquer condição desconhecida for existente. Quando um PEP envia a mensagem solicitando uma política de decisão, há uma consulta na sua base local de configuração para identificar a política de itens que serão avaliadas, nesse momento o PEP passa essa solicitação do item avaliado para o PDP local e incorpora o resultado parcial. O *framework* IETF estipula o COPS para transferência de políticas de decisões para o ponto de aplicação de política e para transferir solicitações do ponto de aplicação de política para o servidor de políticas. O padrão é aberto para outros tipos de mecanismos assim como HTTP, *File Transfer Protocol* (FTP) ou SNMP.

2.2 MONITORAMENTO DE REDES

A gerência de redes se distingue em duas categorias funcionais: monitoramento e controle. O monitoramento é a atividade que faz o acompanhamento de todas as atividades da rede no sentido de contabilizá-los, e o controle é a função que permite que os ajustes sejam feitos visando melhorar o desempenho da rede. O monitoramento enquanto funcionalidade de gerenciamento de rede pode ser relacionado como: ativo e/ou passivo.

O monitoramento passivo avalia os pacotes que estão transitando na rede sem interferir no fluxo de pacotes e consequentemente na performance da rede, produzindo um substancial

volume de dados coletados a qual resultará a base pela qual a investigação se procederá. O grande desafio dessa abordagem é conseguir restringir a quantidade de dados coletados com isso aprovisionando somente as informações fundamentais.

O monitoramento ativo é realizado pelo acréscimo e investigação de pacotes de teste na rede. Nesta técnica a quantidade de dados armazenados não é relevante, em contrapartida, o desafio é adequar o volume de pacotes inseridos para sucessão dos testes e atingimento das métricas desejadas sem que se interfira no desempenho e alocação excessiva de recursos.

Para obter as informações de gerenciamento, os administradores de redes são apoiados por um conjunto de ferramentas integradas utilizadas no gerenciamento de redes (OLIVEIRA et al., 2017).

O monitoramento desenvolve a função de acompanhamento de todas as atividades da rede no sentido de contabilizá-los, com isso possuí dois tipos de monitoramento principais: o monitoramento baseado em captura de pacotes e o monitoramento baseado em fluxos.

O monitoramento baseado em investigação de pacotes é definido como um sistema investigador de pacotes que constantemente pode ser qualificado como *packet sniffing* (coletor de pacotes) ou *protocol analysis* (analisador de protocolo).

Um analisador de pacotes é tipicamente implementado por uma ferramenta coletor de pacotes, que captura os sinais que saem e chegam pela interface de rede e os aprovisiona e analisa, de acordo com a interpretação baseada nos protocolos suportados pela ferramenta.

O monitoramento baseado em investigação de pacotes apresenta vários benefícios, dentre elas: permite um melhor entendimento das características da rede; verificação de quem está ativo na rede; indicar o que está consumindo a largura de banda disponível ou quem; identificar os horários de pico de uso da rede; identificar possíveis atividades maliciosas; detectar aplicações inseguras e infectadas.

O monitoramento do tráfego de rede baseado no conceito de fluxos IP, foi originalmente desenvolvido pela empresa *CISCO Systems*, através do protocolo proprietário *NetFlow* que se tornou um modelo de fato devido a sua ampla adoção pelos dispositivos de diversos fornecedores. Em sua última versão o *NetFlow v9* trouxe o conceito de definição de campos flexíveis baseado em modelos definidos diretamente pelos usuários. Os padrões definidos pelo protocolo *NetFlow*, foram empregados e melhorados pelo protocolo *IPFIX* que tem como objetivo ser o novo padrão comum para o monitoramento de fluxos de tráfego *Internet Protocol* (IP), definido pelo *Internet Engineering Task Force* (IETF). Já o *sFlow* é um padrão no que se refere a monito-

ramento de redes de compuatdores considerado novo. *sFlow* possui especificação (RFC 3176) a sua implementação foi difundida a partir de 2001 com sua primeira versão. O modelo *sFlow* refere-se a um mecanismo para aprisionar dados de tráfego em redes comutadas ou roteadas.

Estas técnicas de monitoramento possuem como premissa o agrupamento do tráfego de rede consolidado em sessões de transmissão unidirecional, denominado como fluxos. Esses fluxos são elaborados em sensores que os encaminham de forma agrupada aos coletores aonde são armazenados e em último plano, habilitam a análise por meio de relatórios.

Os fluxos, também conhecidos como fluxos de tráfego ou fluxo de dados, são conjuntos de tráfego de redes (aplicativos, protocolos e informações de controle) que possuem atributos comuns, como endereços de origem/destino, tipos de informações, sentido, ou informações fim-a-fim.

As informações dentro de um fluxo são transmitidas em uma simples sessão de uma aplicação. Fluxos fim-a-fim, entre origem e destino de aplicativos, dispositivos e usuários. Desde que eles sejam identificados pelas suas informações fim-a-fim, eles podem ser diretamente relacionadas a um sistema, equipamento, rede ou relacionadas com um usuário final. Pode-se também examinar fluxos que transitam por um enlace ou fluxos rede-a-rede básica.

Na academia, as iniciativas de pesquisa a respeito do uso de PBNM com monitoramento de redes organizou-se no viés do uso de parâmetros de QoS, como podemos observar nos trabalhos de AHMED; MEHAOUA; BOUTABA (2002), RIBEIRO (2003), COSTA et al. (2005) e BELLER (2005).

O trabalho de AHMED; MEHAOUA; BOUTABA (2002) propõem um gerenciador de QoS do *DiffServ* com o objetivo de superar as limitações do modelo de gerenciamento de QoS do *DiffServ* estático, onde os monitores de rede são usados para tornar o sistema reativo, tomando decisões automáticas e em tempo real sobre o tráfego fora do perfil. Os autores descrevem a proposta de usar *feedback* de monitoramento de rede e ponto de decisão de política. As informações de monitoramento coletadas são usadas para gerenciar e adaptar dinamicamente parâmetros de QoS para o tráfego do usuário. Esse sistema envolve um sistema de gerenciamento baseado em políticas para obter um comportamento de rede mais dinâmico no tratamento do tráfego de usuários.

Outro trabalho acadêmico que utiliza monitoramento de redes juntamente com PBNM é proposto por RIBEIRO (2003) o qual propõem um sistema e uma definição para monitoramento de QoS no qual os modelos de políticas são informações de entrada do sistema utilizados para

checar a QoS inspecionada. No instante em que é detectado uma degradação na QoS, o sistema de monitoramento notifica um gerente com suporte ao protocolo SNMP empregando mensagens do modelo *InformRequest*. A arquitetura do sistema é dividida internamente em controladores de monitores de QoS e monitores QoS. Cada controlador de monitor QoS controla diversos monitores de QoS, os quais coletam informações da rede. Tais informações são comparadas com políticas traduzidas pelo controlador, e, caso sejam detectadas degradações, o controlador de monitor de QoS notifica o gerente. A comunicação entre controlador de monitores de QoS e monitores de QoS também é baseada em SNMP. O principal objetivo do trabalho é fornecer uma solução que integre monitoramento de QoS e OBNMem um único ambiente de gerenciamento.

No trabalho de COSTA et al. (2005) é proposto um módulo de negociação baseado em políticas para o gerenciamento de grades computacionais, de modo a permitir o controle e o monitoramento dos recursos da grade do Projeto Grad-Giga. O trabalho propõem um módulo de negociação que permitir negociar alguns parâmetros de rede e recursos (como capacidade de processamento, número de máquinas) como base nas informações de gerência de cada site. O módulo é capaz de interagir com a base de informações de contabilização e desempenho do site. O módulo, por ser baseado em políticas, permite a simplificação das operações de rede e do provisionamento do gerenciamento. A simplificação das operações de rede é feita por meio de abstrações de alto nível. Estas abstrações podem ser traduzidas em informações de configuração e políticas que serão armazenadas na máquina que realiza o gerenciamento (estação gerente). As informações de configuração e políticas precisam ser distribuídas aos dispositivos para que estes sejam configurados de acordo com as políticas de baixo nível, ou seja, ações executadas por cada dispositivo. Os objetos a serem gerenciados são: quantidade de memória, velocidade de CPU, carga de processamento e capacidade de disco. Caso um destes objetos não atenda às requisições de uma tarefa a ser executada na grade, o cliente poderá negociar a configuração da grade antes da submissão da tarefa, permitindo a sua execução.

Outro trabalho acadêmico acerca de monitoramento de redes juntamente com PBNM é abordado por BELLER (2005), o qual apresenta uma arquitetura de gerenciamento de redes baseado em políticas para automatizar os processos de geração e distribuição de configuração para dispositivos de rede em um ambiente *DiffServ*. A arquitetura é baseada nos padrões do IETF e introduz um novo modelo de política de alto nível para simplificar o processo de descrição das políticas de QoS. A arquitetura é definida em três camadas: modelo de política de alto nível, modelo de política independente de dispositivo e um modelo de política dependente

de dispositivo. O trabalho explica os modelos propostos e descreve os processos utilizados para conversão das políticas de alto nível em configuração de dispositivos. Dentro da arquitetura, também são discutidas e avaliadas três diferentes estratégias para o servidor de políticas atualizar dinamicamente a configuração dos dispositivos de rede.

2.3 *BIG DATA*

A expressão *Big Data* foi idealizada nos anos 2000, com a expansão informacional das ciências como a genômica e a astronomia. Essa expressão, relaciona-se aos grandes volumes de dados, a primeira vez que foi citado foi no periódico britânico *The Economist*, com título *"Data, data, everywhere: a special report on managing information"*. No entanto, durante estes anos o termo foi sendo empregado e relacionado a soluções de *business intelligence* (BI), com *data sets* de terabytes de dados, ou *datawarehouses*. O fato é que *Big Data* simboliza muito mais que isto, e modernamente, o conceito está se transferindo para todos os campos do conhecimento humano (FACHINELLI, 2014).

A definição de *Big Data* é apresentada no relatório *"Data: the next frontier for innovation, competition, and productivity"*, como "a coleção de dados cujo tamanho vai além da capacidade para armazenar, capturar, analisar e gerenciar de ferramentas de software de banco de dados convencionais"(MANYIKA et al., 2011).

O termo *Big Data* está apoiada em três fatores de sustentação, denominado 3 V's do *Big Data*: Volume, Variedade e Velocidade (JUNIOR et al., 2016). O volume refere-se ao fato de que a quantidade de dados disponível cresce de forma exponencial. A variedade diz respeito aos dados serem ou não estruturados. E a velocidade que os dados são capazes de serem capturados e analisados em tempo real (BRETERNITZ; SILVA, 2013).

As tecnologias de *Big Data* estão se alongando em direção a todos os domínios da engenharia e ciência, incluindo medicina, física e biologia. Estas tecnologias demostram que a gerência de grandes volumes de dados está além da capacidade que dispositivos de software conseguem analisar e armazenar estes dados em um espaço de tempo aceitável. O desafio principal para os sistemas de Big Data é investigar os grandes volumes de dados e com isso tirar proveito do conhecimento para futuras ações ou informações úteis (LESKOVEC; RAJARAMAN; ULLMAN, 2014).

Na academia, as iniciativas de pesquisa a respeito do uso de *Big Data* juntamente com correlações de eventos incluem, por exemplo, o trabalho realizado por ZAMAN et al. (2015),

no qual o autor desenvolve um *framework* a partir do *Apache Storm* para filtrar eventos com alto número de inter-relacionamento. O *framework* proposto foi projetado para correlacionar eventos usando o *Storm*, os eventos correlacionados que geram alarmes são usados para solucionar falhas na rede. Este Sistema é composto por dois componentes principais, sendo eles, um balanceador de carga, e um redutor de dados. O balanceador de carga atua como um funil no fluxo de dados vindos dos elementos de redes (*Network Elements* - NEs) na entrada do *framework*. Uma vez que os dados são carregados, o redutor de dados abstrai o fluxo de rastreamento através de um localizador em relação à base de métricas e reduz o tamanho do fluxo filtrando eventos inter-relacionados e, assim, gerando alarmes que administradores de rede sejam capazes de solucionar incidentes. Esse trabalho é limitado pois aplica somente um método de correlacionamento de dados, também não proporcionando ao administrador cadastrar novos tipos de correlação, além de somente ser aplicado em fluxos de rede de telefonia móvel, com isso, reduzindo o uso do sistema pelo administrador de rede.

Outro trabalho acadêmico acerca de *Big Data* e correlações de eventos, é abordado por CHENG et al. (2011), onde é apresentado um sistema de monitoramento baseado em Processamento de Eventos Complexos (*Complex Event Processing* - CEP) para detecção de anomalias de rede. A arquitetura do sistema é composta por três componentes principais, sendo eles, um coletor de dados de evento, um registro de consultas, e um construtor de monitoramento de rede. O funcionamento do sistema se procede de forma que, o *Event Data Collector* coleta os dados de serviços de rede, já o *Query Register* registra as consultas definidas pelo administrador de rede e os tipos de eventos correlacionados, por fim o *Monitoring Net Constructor* o qual é o núcleo do sistema, constrói uma rede de monitoramento que une os dados coletados dos serviços de rede e as consultas definidas pelos administradores de rede. Esse componente faz com que os dados sejam correlacionados e ao seu fim seja gerado uma saída, tal como, um alerta, uma mensagem ou uma ação para os serviços de rede. O sistema apresentado pelos autores para monitoramento de anomalias de serviços de rede em tempo real, utiliza linguagem *Structured Query Language* SQL-like para efetuar o correlacionamento dos eventos. O sistema acaba se limitando, pois o administrador de rede deve possuir domínio da linguagem SQL-like e dos fluxos analisados para conseguir produzir correlações de dados que possam ser proveitosas na análise dos dados.

No trabalho de ROBITZSCH et al. (2015), foi proposto um *framework* que analisa de forma automática os arquivos de registros no *Operations Support Systems* (OSS) a respeito

dos eventos que precedem incidentes de rede. Essa análise é usada a fim de tentar avaliar o motivo que gerou o incidente. Logo após a análise é gerado uma lista ordenada das *Candidate Corrective Action* (CCAs), a qual deve ser apresentada a um especialista de domínio no Centro de Operações de Rede (*Network Operations Center* - NOC) indicando possíveis soluções para um incidente detectado. O sistema possuí cinco módulos, sendo eles, *Dimension Reduction Module* (DRM), *Episode Discovery Module* (EDM), *Episode Classification Module* (ECM), *Pattern Matching Module* (PMM) e *Recommender System Module* (RSM). O sistema funciona de forma que, o *Dimension Reduction Module* (DRM) cria uma representação unidimensional do fluxo de dados, além de detectar e remover eventos que podem ser considerados ruídos, o EDM realiza a análise do fluxo de dados afim de encontrar sequencias fechadas de eventos, o ECM classifica os EDM em padrões, assim alimentando o *Pattern Module Libery* (PML) o qual mantem todos os padrões. Uma vez que o PML é preenchido com sucesso o PMM procura o fluxo de dados de entrada para padrões conhecidos, por último, o RSM recebe os padrões combinados e previstos a partir PMM e recomenda CCAs para os especialistas de domínio no NOC. O sistema proposto demanda um conhecimento grande dos dados e fluxos de rede que estão sendo analisados, para que o administrador consiga optar por uma ação correta a ser tomada para cada incidente. Já que o sistema apenas propõem possíveis correções com base na análise realizada.

2.4 SÍNTESE

Neste capítulo, foram analisados sete trabalhos que utilizam monitoramento de redes juntamente com PBNM ou que realizam o monitoramento de redes em ambientes *Big Data*. Em AHMED; MEHAOUA; BOUTABA (2002), RIBEIRO (2003), COSTA et al. (2005) e BELLER (2005), são apresentados conceitos de PBNM para utilização no monitoramento de redes levando em consideração apenas parâmetros de configuração e provimento de QoS.

Nos trabalhos ZAMAN et al. (2015), CHENG et al. (2011) e ROBITZSCH et al. (2015), que realizam monitoramento de redes em ambientes *Big Data* acabam por limitar-se, pois aplicam o monitoramento de redes em fluxos de redes específicos reduzindo o uso do sistema pelo administrador de redes, além de também utilizar linguagens de baixo nível para realizar o monitoramento de redes e com isso obrigando o administrador a possuir um conhecimento grande dos dados e fluxos de rede que estão sendo analisados, para que com isso consiga optar por uma ação correta a ser tomada para cada incidente, já que os sistemas apenas propõem possíveis

correções com base nas análises realizadas.

A arquitetura proposta nesta dissertação lave em consideração diversos tópicos presente nos diferentes trabalhos discutidos neste capítulo, como uso no monitoramento de redes de PBNM em um ambiente *Big Data*, pois ao analisar estes trabalhos verificou-se que ao utilizar PBNM, o monitoramento de redes fica mais próximo do administrador de redes visto que o monitoramento é descrito em alto nível. Já através da leitura desses mesmos trabalhos os quais utilizavam técnicas de *Big Data* no monitoramento de redes verificou-se que administrador possuí uma capacidade mais reativa uma vez que o processamento dos dados ou fluxos podem ser realizados em tempo real e também não sendo afetado pelo aumento de dados pois técnicas de *Big Data* também trabalham com paralelismo.

3 SOLUÇÃO PROPOSTA

Como apresentado anteriormente, o uso de PBNM no monitoramento de redes a fim de controlar uma combinação de equipamentos baseados em diferentes tecnologias e também não só usar PBNM com a finalidade de controlar e aplicar qualidade de serviço (*Quality of Service* - QoS) é um tema pouco explorado pela comunidade de gerenciamento. Não sabe-se como essas abordagens irão se comportar quando empregados neste contexto. Com objetivo de responder a esta pergunta, foi definida uma arquitetura de monitoramento de redes baseado em políticas em um ambiente *Big Data*. Sendo assim, neste capítulo serão apresentados a arquitetura, os componentes e o funcionamento da mesma, além de demonstrar a estrutura da política desenvolvida para o monitoramento.

3.1 ARQUITETURA

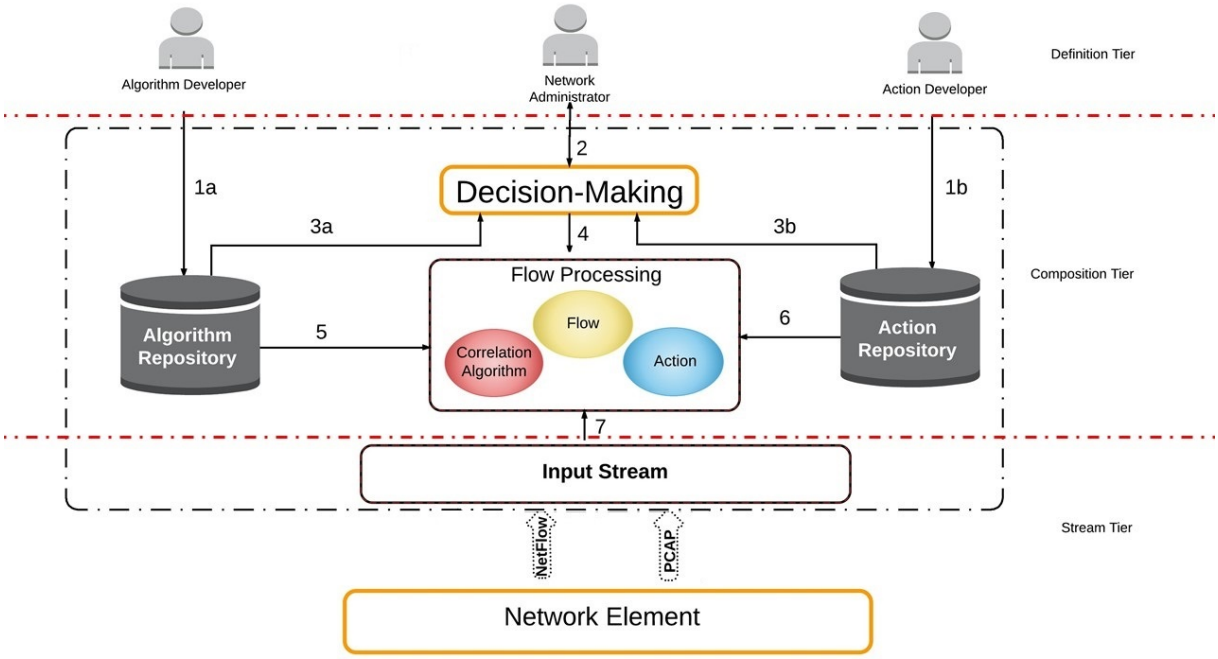
A arquitetura proposta neste trabalho, está dividida em três camadas. Esta divisão, possui como principal vantagem possibilitar uma melhor estruturação dos elementos funcionais da arquitetura. Com isso, foi desenvolvido um sistema de monitoramento de redes, que possibilita a execução de algoritmos de correlação de dados e ação, permitindo ao administrador o uso dos mesmos, através de políticas de gerenciamento de redes.

Na arquitetura, são definidos três tipos de operadores humanos que possuem papéis diferentes: os administradores, os desenvolvedores de algoritmo e os desenvolvedores de ações. Em todos os casos, a interação com o sistema ocorre através do terminal de comando. O administrador é o operador humano interessado em monitorar a rede. O desenvolvedor de algoritmo por sua vez, tem como função definir os algoritmos que farão as correlações de dados. Já o desenvolvedor de ação tem como objetivo a criação de algoritmo de ações a serem tomadas após a análise dos dados. De fato, um mesmo desenvolvedor de algoritmo ou de ação pode assumir os dois papéis, porém, funcionalmente, são papéis distintos.

A primeira camada da arquitetura é onde estão os operadores humanos, na segunda onde é feito o processamento do sistema, e por fim na terceira onde são captados os fluxos de rede. A seguir são apresentados na Figura 3 as camadas sumarizadas:

a) *Definition Tier* é a camada onde estão localizados os desenvolvedores de algoritmos e o administrador. Os desenvolvedores criam novos algoritmos e cadastram suas descrições.

Figura 3 – Arquitetura da Solução da Proposta



Fonte: acervo pessoal.

O administrador utiliza os algoritmos prontos através de políticas que expressam seus interesses.

b) *Composition Tier* é a camada onde estão localizados os elementos em que serão aplicadas as definições contidas na política, a qual foi definida pelo administrador. Nessa camada existe um módulo de processamento de fluxo, chamado *Flow Processing*, o qual possui como função principal, analisar os fluxos de rede injetados no sistema e ao final do processamento gerar uma ação esperada pelo administrador.

c) *Stream Tier* é a camada onde está localizado o gerenciador de fluxos de dados, kafka, o qual é responsável por disponibilizar os fluxos de rede cadastrados no mesmo.

3.1.1 Componentes da Arquitetura

A arquitetura do sistema é composta por cinco componentes principais, descritos a seguir:

a) *Decision-Making*: Responsável por tomar decisões sobre políticas, atua como Ponto de Decisão Política (*Policy Decision Point*);

b) *Algorithm Repository*: Repositório onde são armazenados os algoritmos produzidos pelos *Algorithm Developers*;

c) *Action Repository*: Repositório onde estão armazenadas as possíveis ações do sistema;

d) *Input Stream*: Responsável por fornecer os fluxos de rede que serão consumidos pelos algoritmos de correlação de eventos;

e) *Flow Processing*: Ambiente *Big Data* onde se encontram todos os *Bolts e Spouts* de correlação de dados e os *Bolts* de ações a serem disparadas após o processamento dos fluxos de rede.

3.1.2 Funcionamento da arquitetura

Baseado na Figura 3 é possível visualizar o funcionamento do sistema. O *Algorithm Developer* cria os algoritmos de correlação de dados e cadastra (1a) os mesmos no sistema; Já o *Action Developers* cria as ações e cadastra (1b) as mesmas no sistema. O *Network Administrator* envia uma requisição (2) para o *Decision-Making* expressando a política a ser implementada, a qual contêm o fluxo, o tipo de entrada e qual processamento deseja gerenciar no sistema; com essas informações o *Decision-Making* aplica a política nos repositórios (3a) (3b) e determina qual Algoritmo de Analise de dados e Ação irão ser usados (4). Por fim, o módulo, *Flow Processing*, recebe o algoritmo de correlação (5), a ação (6) e o tipo de fluxo requisitado (7) pelo administrador de rede. Após a associação destes componentes é feito o processamento dos dados e gerado uma saída.

Na subseção seguinte, será demonstrado o funcionamento e a estrutura da política baseada em gerenciamento de redes na arquitetura. Será possível observar como são desenvolvidas as políticas para aplicação e implantação no sistema.

3.2 PBNM

A especificação da política para aplicação no sistema é definida utilizando arquivo de configuração *(eXtensible Markup Language* (XML) (BRAY et al., 2008). Pois o arquivo XML permite alterações no modelo da informação, bem como a adição de novos atributos na política sem a necessidade de modificação nos mecanismos do sistema. O padrão XML foi escolhido devido à sua ampla aceitação para especificação de configuração de sistemas e por ser um

padrão mais genérico, porém as especificações das *tags* contidas no arquivo XML criado para está dissertação são baseados nos trabalhos de DAMIANOU et al. (2001) e KRYFTIS et al. (2017).

Na Figura 4 é demonstrado o modelo de política em XML que é utilizado para ser preenchido pelo administrador e aplicado no sistema.

Figura 4 – Modelo de especificação de política

```
<politicsConf>
        <confID>      </confID>
        <basicInfo>
                <trigger>             </trigger>
                <correlation>     </correlation>
                <correlatType> </correlatType>
                <action>              </action>
        </basicInfo>
        <correlationInfo>
                 <checkList>
                        <ipAdress>
                            <source> </source>
                            <dest>      </dest>
                        </ipAdress>
                        <protocol> </protocol>
                        <service>    </service>
                        <port>          </port>
                        <timeMSAmount> </timeMSAmount>
                        <eventsAmount> </eventsAmount>
                </checkList>
        </correlationInfo>
        <actionInfo>
                <logLocal>          </logLocal>
                <email>               </email>
        </actionInfo>
</politicsConf>
```

Fonte: acervo pessoal.

Neste arquivo são especificados quatro elementos principais, os quais estão marcados por cores diferentes para melhor explicação. Os quatro elementos, são:

a) $<$ *confID* $>$ marcado em verde;

b) $<$ *basicInfo* $>$ em vermelho;

c) $<$ *correlationInfo* $>$ em amarelo;

d) $<$ *actionInfo* $>$ em azul.

O elemento < *confID* > é a *tag* utilizada para nomear a política criada pelo administrador, e tem seu preenchimento obrigatório. Já o elemento < *basicInfo* > contém os principais comandos utilizados para que o módulo *Decision-Making* que atua como um *Policy Decision Point*, explicado na subseção 4.2.1, seja capaz de aplicar corretamente a política especificada no sistema. Este elemento também tem seu preenchimento total obrigatório, para o pleno funcionamento do sistema. A *tag* < *basicInfo* > possui quatro *tags* vinculadas.

a) < *trigger* > indica qual o gatilho utilizado para desencadear a ação após a aplicação da política (*i.e.,* gatilho de tempo, a cada X milissegundos uma ação é disparada ou gatilho de evento, a cada Y número de fluxos de dados analisados uma ação é disparada);

b) < *correlation* > define o tipo de correlação que deve ser usado na analise dos fluxos de dados (*e.g.,* filtragem, compressão, contagem, *etc.*);

c) < *correlationType* > designa de que modo os fluxos injetados de ferramentas diferentes irão ser modelados (*i.e,* fluxos provenientes de *netflow* são diferentes de fluxos de um *sniffer*, esses fluxos precisam ter um formato idêntico para poderem ser processados pelo sistema);

d) < *action* > especifica o tipo de ação a ser tomado após o processamento dos fluxos (*e.g.,* geração de um *log*, *script* de autoreconfiguração).

O elemento < *correlationInfo* > compreende os dados utilizados pela *tag* < *correlation* > presente no < *basicInfo* > para realizar o processamento dos dados. Só um campo nesse elemento possue preenchimento obrigatório, ou o < *timeMSAmount* > caso a *tag* < *trigger* > presente no < *basicInfo* > seja preenchida com um gatilho de tempo, ou o < *eventsAmount* > caso a *tag* < *trigger* > seja preenchida com um gatilho por número de eventos analisados. A *tag* < *correlationInfo* > também possui outras *tags* subordinadas a ela, são elas:

a) < *ipAdress* > é uma *tag* utilizada para cadastrar o endereço ip dos fluxos a serem monitorados e possui outras duas *tags* auxiliares;

   - < *source* > indica o ip de origem do fluxo;
   - < *dest* >indica o ip de destino do fluxo.

b) < *protocol* > usada com o intuito de monitorar fluxos com protocolos específicos (*e.g.,* TCP);

c) $<$ *service* $>$ utilizada para monitorar fluxos com serviços exclusivos (*e.g.,* HTTP, SMTP, SSH, *etc.*);

d) $<$ *port* $>$ indica a porta específica monitorada (*i.e.,* se o administrador deseja monitorar transferência de arquivos, irá monitorar a porta 21);

e) $<$ *timeMSAmount* $>$ indica o tempo máximo que cada conjunto de fluxos será processado até a geração de uma nova ação;

f) $<$ *eventsAmount* $>$ indica o numero máximo de fluxos analisados por geração de ação.

Por fim o elemento $<$ *actionInfo* $>$ é usado para designar o tipo de ação que deverá ser aplicada pelo sistema ao final do processamento dos dados. Também possui relação com o elemento $<$ *basicInfo* $>$, porém mais especificamente a uma *tag* presente nesse elemento, $<$ *action* $>$, que indicará o tipo de ação (*e.g.,* geração de um *log*, envio de um *e-mail* para o administrador, *script* de autoreconfiguração). É um elemento que necessita que pelo menos uma das *tags* seja preenchida, ou $<$ *logLocal* $>$ ou $<$ *email* $>$.

4 IMPLEMENTAÇÃO

Neste capítulo serão apresentados detalhes de implementação do protótipo da arquitetura de gerenciamento apresentada no capítulo 3. Além de apresentar as tecnologias utilizadas e os módulos presentes na arquitetura proposta.

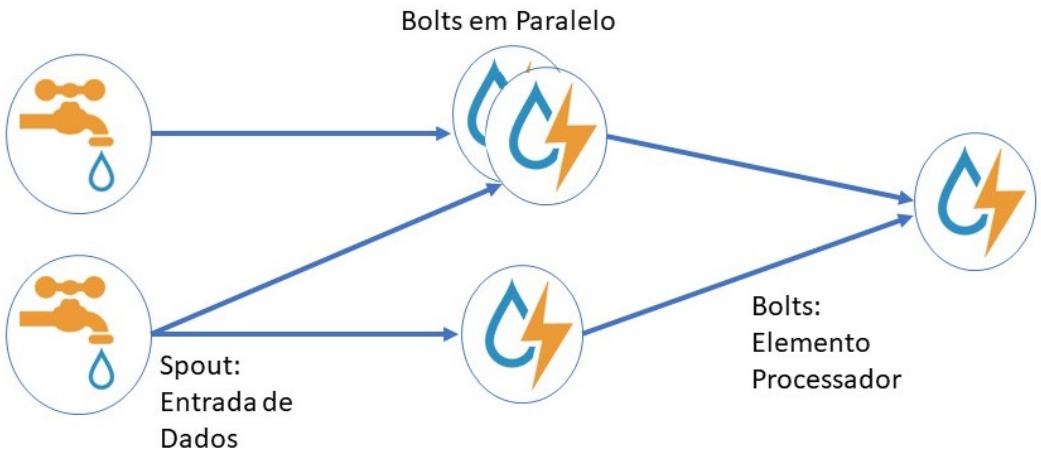
4.1 *APACHE STORM*

Apache Storm é o *framework* utilizado nesta dissertação, pois é um sistema de processamento de fluxo de dados distribuído (*Data Streaming Processor* - DSP) e em tempo real com tolerância a falhas (TOSHNIWAL et al., 2014), o que garante o processamento eficiente de um fluxo continuo e ilimitado de dados. Também é um projeto *open-source* licenciado sob a licença EPL - *(Eclipse Public License)*. A EPL é uma licença muito permissiva, possibilitando que o projeto seja utilizado para qualquer fim, sendo proprietário ou *open-source*.

Com a capacidade de processar fluxos de dados ilimitados de maneira simples e transparente, o *Apache Storm* faz para o processamento em tempo real o que o *Hadoop* fez para o processamento em lote, além de possuir uma arquitetura descomplicada, podendo ser usado com qualquer linguagem de programação. O *framework* implementa um modelo no qual os dados fluem continuamente através de uma rede de estruturas de transformação. A abstração de um fluxo de dados é chamada de *Stream* que é uma sequência ilimitadas de tuplas. Uma tupla é uma estrutura que pode representar tipos padronizados de dados (*e.g.,* inteiros, pontos flutuantes, *etc*) ou tipos definidos pelo usuário através do uso de algum código. Cada fluxo é definido por uma identificação única que pode ser usada para construir topologias de fontes de dados. Os *streams* originam-se de estruturas chamadas *Spouts*. A estrutura responsável por consumir e produzir os dados do fluxo é chamado de *Bolt*.

O modelo conceitual desta topologia de processamento de fluxo de dados do Apache Storm é apresentado na Figura 5. Uma topologia é distribuída entre processos de trabalho, onde para cada componente, *Spout ou Bolt*, é atribuído um número definido de tarefas a serem distribuídos entre os processos de trabalho. Atualmente, o *Apache Storm* é utilizado como solução para processamento de fluxo de dados de *Big Data* por grandes empresas.

Figura 5 – Topologia do Storm



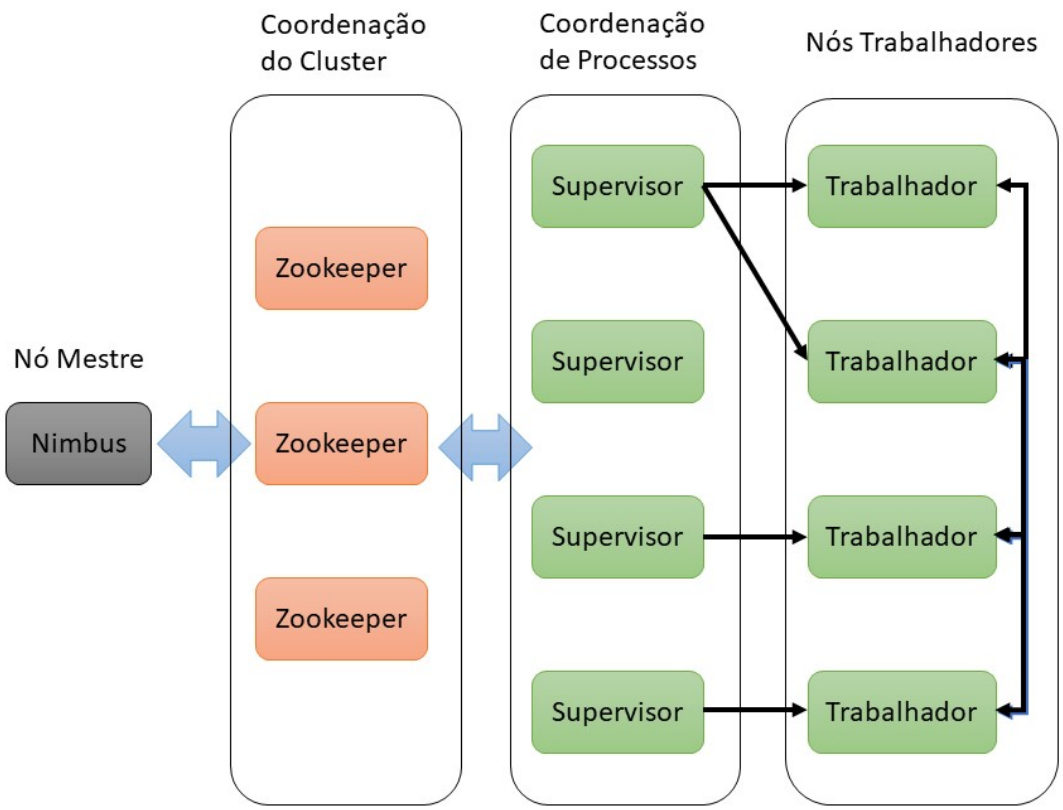
Fonte: acervo pessoal.

A Figura 6 expressa como é feita a coordenação de processos no *Storm*. Existem dois tipos de nós em um *cluster Storm*, o mestre e os nós trabalhadores. O nó mestre, executa o *Nimbus*, que recebe as informações da topologia, definidas pela aplicação programada pelo usuário, e coordena a instanciação de cada processo necessário para atender a especificação da topologia, ou seja, o *Nimbus* coordena a instanciação dos *spouts e bolts*. Esses processos executam nos nós trabalhadores. Cada nó trabalhador executa um *daemon* chamado *supervisor*, que monitora os processos da topologia e informa o estado para o *Zookeeper*. O *Zookeeper* possui então uma visão geral do estado da topologia e repassa essa informação para o *Nimbus* que pode decidir realocar processos, como no caso de falha por exemplo.

4.2 PROTOTIPAÇÃO

Baseado na arquitetura proposta no capitulo 3, foi desenvolvido um sistema baseado no *Apache Storm*, a função desse sistema é permitir ao administrador a criação de uma política específica para correlacionar eventos de fluxos de rede, e com isso realizar a gestão da mesma. O gerenciamento de redes baseado em políticas tem se mostrado uma técnica eficaz na definição de parâmetros de utilização e comportamento esperado das redes (SLOMAN, 1994). Além disso, graças ao fato de existirem dois repositórios de algoritmo, o administrador tem a liberdade de

Figura 6 – Coordenação de processos no Storm



Fonte: acervo pessoal.

escolher com quais algoritmos irá trabalhar.

O protótipo desenvolvido contém dois desenvolvedores no sistema, cada um é responsável por cadastrar novos algoritmos de correlação e ação além disponibilizar um arquivo XML contendo as informações das possíveis funcionalidades e características de cada algoritmo. A seguir são discutidos e avaliados os módulos dos sistemas, onde são apresentados as características especificas do sistema e é demonstrado como cada módulo se comporta quando é aplicada a política

4.2.1 Módulo *Decision-Making*

O módulo *Decision-Making* atua como o *Policy Decision Point* do sistema, a política definida pelo administrador é interpretada e aplicada nos outros módulos, que são, o de correlacionamento, o de fluxos e o de ações. De acordo com a tradução da política feita pelo

Decision-Making são enviadas ações de configuração para os módulos subordinados.

O modelo de política que o administrador usará no sistema pode ser revista na seção 3.2.

4.2.2 Módulo Correlacionador

Neste módulo existem dois repositórios, um para algoritmos de correlação de dados e outro para algoritmos de ações. No repositório de correlação de dados são armazenados todos os algoritmos de correlação e a descrição dos mesmos. Sendo possível a criação de qualquer método ou algoritmo relacionado a correlação de eventos. Os tipos de correlação de eventos que podem ser implementados em forma de *Bolts*, segundo JAKOBSON; WEISSMAN (1993), são: Compressão; Supressão seletiva; Filtragem; Contagem; Escalação; Generalização; Especialização; Aglutinação; Além disso, encontra-se alguns métodos segundo os mesmos autores, o qual também podem ser implementados e inseridos no sistema, por exemplo, Correlação Baseada em Regras, Correlação utilizando Lógica Difusa, Redes Bayesianas ou Redes Causais, Raciocínio Baseado em Modelos, Filtragem, etc.

O módulo correlacionador é responsável, como o próprio nome refere-se, por correlacionar os fluxos de dados processados pelo sistema, e também por limitar o número de eventos analisados e gerar ações após o processamento. Este módulo desempenha a função de *Policy Enforcement Point* no ponto de vista de um sistema de PBNM.

A Figura 7 demonstra o funcionamento do módulo correlacionador na arquitetura descrita no capítulo 3. O elemento conhecido como *Spout*, é responsável por injetar todos os fluxos de rede selecionados pelos *Bolts* de correlação. Já os elementos *Bolts* de fluxo, são responsáveis por capturar um tipo especifico de fluxo de rede (*e. g.,* pcap, *netflow* ou qualquer outro tipo de fluxo) previamente programado no *Bolt*.

Os *Bolts* de fluxo implementados no sistema, são:

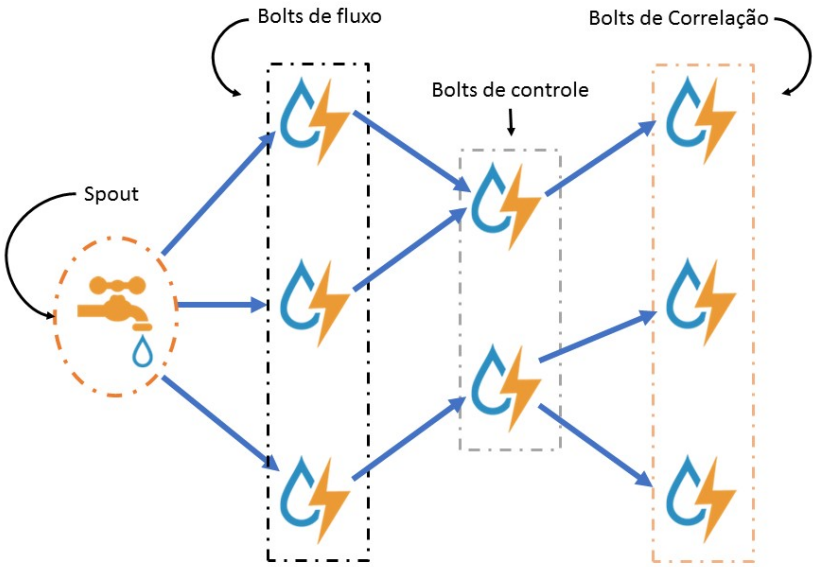
a) *PcapBolt*, separa fluxos injetados no sistema provenientes de *sniffer* de rede.

b) *NetflowBolt*, destaca fluxos inseridos no sistema provenientes de coletores *netflows*.

Os *Bolts* de controle, são onde ocorrem as limitações de processamento de fluxo de rede injetados no sistema, que pode ser por tempo ou por número eventos correlacionados, ou qualquer outro programado.

Os *Bolts* desenvolvidos para este sistema foram:

Figura 7 – Módulo correlacionador no *Apache Storm*



Fonte: acervo pessoal.

a) *ChronoBolt*, o qual dispara uma ação esperada pelo administrador a cada tempo pré programado.

b) *EventBolt*, dispara uma ação a cada número de eventos analisados em um fluxo de rede.

Os *Bolts* de correlação são responsáveis por efetuar a correlação de eventos nos fluxos selecionados.

Os *Bolts* correlacionadores desenvolvidos foram:

a) *CompressionBolt*, que comprime pacotes de dados cadastrados pelo administrador.

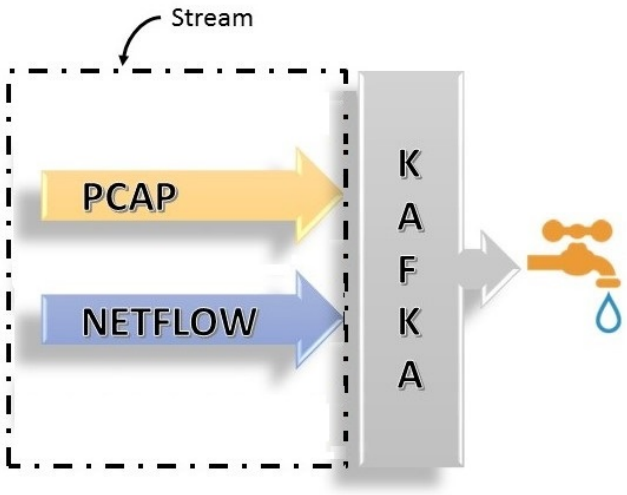
b) *CounterBolt*, este *bolt* conta o pacotes cadastrados pelo administrador.

c) *FilterBolt*, este *bolt* filtra somente os pacotes cadastrados pelo administrador.

4.2.3 Módulo Fluxos

Este módulo disponibiliza os fluxos de rede para o sistema. A Figura 8 demonstra o funcionamento do módulo.

Figura 8 – Módulo de fluxos no *Apache Storm*



Fonte: acervo pessoal.

Como pode ser visualizado na Figura 8 o elemento *KAFKA* que é responsável pela aglutinação dos fluxos de diferentes fontes, em um único tópico do *KAFKA* (*i.e.* quando o sistema receber fluxos de *SNIFFER* de rede, pacotes .pcap ou quando receber um fluxo de *NETFLOW* o sistema irá destinar os fluxos a um único tópico específico no *Kafka*) com isso disponibilizando esses diferentes fluxos para o sistema.

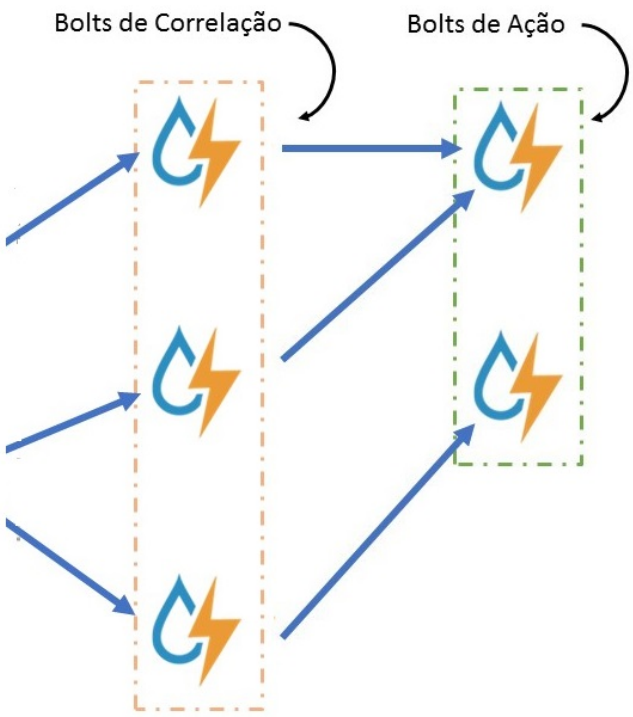
O módulo necessita de uma pré configuração, após esta configuração é executado um *script* em *JAVA* que tem a função de capturar os fluxos dos tópicos já mencionados e inseri-los nos *Spouts*.

4.2.4 Módulo Ações

Este módulo é responsável por executar o algoritmo gerador da ação. A Figura 9 demonstra que ao final do processamento do *Bolt* de correlação é ativado um gatilho que desperta o *Bolt* de ação para que seja gerada a ação esperada pelo administrador de rede.

Após a implementação dos módulos descritos neste capítulo foram realizados teste a fim de validar a arquitetura proposta que serão discutidos no capitulo 5 onde é possível compreender o autêntico comportamento do sistema.

Figura 9 – Módulo de ações no *Apache Storm*



Fonte: acervo pessoal.

4.3 SÍNTESE

No presente capítulo foi detalhada a proposta de implementação para a arquitetura de monitoramento baseado em políticas apresentada no capítulo 3. Nesta implementação, optou-se pelo uso de arquivos XML para definições de política, principalmente, devido a sua estrutura de representação que possibilita a assimilação intuitiva das informações durante a leitura humana. Esta estratégia, inclusive, facilita na interpretação das políticas definidas pelo administrado pelo módulo *Decicion-Making* que atua como *Policy Decision Point* segundo as diretrizes do IETF e DMTF. Neste capítulo também foram apresentados os módulos que atuam como *Policy Enforcement Point* segundo as diretrizes do IETF e DMT, que são: o módulo correlacionador, módulo de fluxos e módulo de ação.

5 VALIDAÇÃO DA SOLUÇÃO

Após a implementação da arquitetura proposta, com o objetivo de demonstrar a utilização da política definida juntamente com os algoritmos de correlação de dados em um ambiente *Big Data* e avaliar a flexibilidade dos modelos propostos para representação de políticas, é apresentado dois estudos de caso para monitoramento de rede em um ambiente realístico.

Foram definidos dois estudo de caso para avaliar os diferentes preenchimento das políticas e o comportamento delas no sistema. No estudo de caso 1, descoberta de rede, o administrador tem como objetivo realizar a verificação de novos dispositivos encontrados na rede gerenciada. Já no estudo de caso 2, descoberta de falha em servidor *Web*, o administrador tem como objetivo verificar a disponibilidade de um servidor *Web* monitorado.

Foi desenvolvido uma topologia de rede com a finalidade de gerar um ambiente realístico. O cenário para validação dos estudos de caso é o da Figura 10, as políticas usadas foram baseadas no modelo de política apresentado na seção 3.2. Primeiramente a política é interpretada pelo sistema que dispara comandos para os módulos subordinados ao módulo *Decision-Making*.

5.1 AMBIENTE DE TESTE

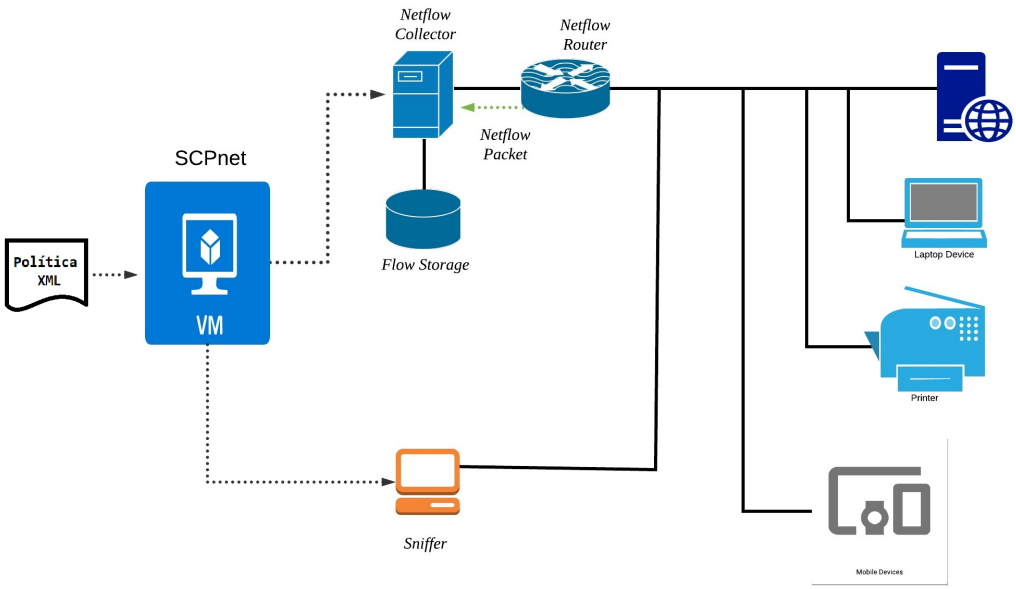
Para os testes de validação da arquitetura, foi utilizado uma máquina em rede, cuja configuração é descria na Tabela 1. A criação da topologia de rede foi realizada utilizando o *software* GNS3.

Tabela 1 – Configuração da máquina utilizada para validação da solução

Processador	Intel Core i7 2.50GHz
Memória RAM	8,00 GB
Sistema Operacional	Windows 10 Home
Máquina Virtual	Oracle VM VirtualBox 5.2

Para a validação da arquitetura, foi utilizado a topologia descrita na Figura 10. Nesse ambiente, os fluxos provenientes de ferramentas diferentes, coletor *NetFlow* e *Sniffer*, são disponibilizados ao sistema pelo módulo de fluxos. O sistema irá interpretar a política, a qual definirá quais os tipos de fluxos serão processados, em qual tipo de gatilho de ação e utilizando qual tipo de técnica de correlação de dados.

Figura 10 – Cenário para validação de estudo de caso



Fonte: acervo pessoal.

5.2 DESCOBERTA DE REDE

A gerência de configuração é responsável por manter um inventário atualizado e produzir relatórios baseados nesse inventário. Com o intuíto de validar este estudo de caso, foi mantido uma rede de computadores básica com um computador, um servidor *Web*, e uma impressora. Este foi o inventário inicial, a fim de comprovar a viabilidade do protótipo foi inserido fluxos de novos dispositivos na rede com isso gerando novos inventários através de *logs* de rede.

Neste estudo de caso, o objetivo é avaliar o efeito da política aplicada no sistema que realiza correlação de dados em um ambiente *Big Data* para realizar uma descoberta de rede passiva sob a perspectiva do administrador de rede e manter um inventário dos dispositivos conectados a rede atualizado. A simulação foi realizada tendo como base o cenário, apresentado na Figura 10. A política utilizada para realizar descoberta de rede é exposta na Figura 11.

Figura 11 – Política usada para descoberta de rede

```
<politicsConf>
        <confID>CompressTest</confID>
        <basicInfo>
                <trigger>ChronoBolt</trigger>
                <correlation>CompressBolt</correlation>
                <correlatType>ProtocolServiceBolt</correlatType>
                <action>LogBolt</action>
        </basicInfo>
        <correlationInfo>
                <checkList>
                        <ipAdress>
                                <source> </source>
                                <dest>      </dest>
                        </ipAdress>
                        <protocol>      </protocol>
                        <service>       </service>
                        <port>              </port>
                        <timeMSAmount>7000</timeMSAmount>
                        <eventsAmount>5</eventsAmount>
                </checkList>
        </correlationInfo>
        <actionInfo>
                <logLocal>/home/storm/StormInfrastructure
                /Storm/apache-storm-1.0.3/examples
                /storm-starter/src/jvm/storm
                /starter/Log</logLocal>
                <email>storm@storm.com.br</email>
        </actionInfo>
</politicsConf>
```

Fonte: acervo pessoal.

O monitoramento de descoberta de rede será realizado a partir da aplicação da política representada na Figura 11. Primeiramente o sistema irá executar a nova política de nome *CompressTest*. Logo após o módulo *Decision-Making* irá acessar os principais comandos, $<$ *trigger* $>$, $<$ *correlation* $>$, $<$ *correlationType* $>$ e $<$ *action* $>$ para realizar o processamento dos fluxos de dados e a geração de uma ação esperada pelo administrador.

Após o acesso aos comando principais, o módulo *Decision-Making* irá traduzir a *tag* $<$ *correlationType* $>$ para um método interno do sistema, especificando o tipo de formatação necessária dos fluxos de dados para posterior processamento destes fluxos. A formatação de fluxos de dados pelo método *ProtocolServiceBolt* modela os fluxos de diferentes ferramentas em um formato idêntico, tornando-os disponíveis para analise do sistema.

Logo após o módulo irá traduzir a *tag* $<$ *trigger* $>$ que irá identificar o gatilho da ação, que nesse caso é por tempo (*i.e,* a cada X milissegundos de processamento será desencadeado uma ação).

Após a definição do modelo de formatação dos fluxos e do gatilho da ação, o módulo interpretará a *tag* $<$ *correlation* $>$ preenchida com o comando *CompressBolt*, que especifica o tipo de correlação usada, que nesse caso é compressão de dados. O *Decision-Making* irá selecionar o Algoritmo 1 de compressão de dados, e determinará quais fluxos devem ser analisados, tomando com base o elemento $<$ *correlationInfo* $>$.

Algoritmo 1: *Bolt* DE COMPRESSÃO

```
1  compressMap : mapeamento de IPs dos pacotes recebidos
2  compressPckts : pacotes mapeados
3  begin
4      //Inicializacao
5      compressMap = StringArray()
6      compressPckts = PacketHash()
7      Evento e disparado quando um pacote p chega :
8      if compressMap.CONTEM(p.IP) then
9          compressPckts.INSERE(p.IP, p)
10     end
11     else
12         compressMap.INSERE(p.IP)
13         compressPckts.INICIA(p.IP)
14         compressPckts.INSERE(p.IP, p)
15     end
16
17     Evento e disparado quando um sinal de acao s chega :
18     GerarXMLCorrelacao(copressMap, compressPckts)
19     EnviarEventoAcao()
20 end
```

O Algoritmo 1 analisará somente os fluxos de rede que possuírem os campos preenchidos, como a *tag* $<$ *timeMSAmount* $>$ que define nessa política um tempo de 7000 milissegundos para cada analise de fluxos e posterior geração de uma ação; Os campos deixados em branco são considerados como vazios e não são levados em consideração na analise dos fluxos de rede.

Com base nos campos da *tag* $< correlationInfo >$ o Algoritmo 1 irá captar os fluxos de dados formatados do sistema, depois comprimirá os dados, com isso gerando um arquivo XML com uma lista de dados não repetidos a cada intervalo de tempo definido na *tag* $< correlationInfo >$.

O módulo *Decision-Making* interpretará a *tag* $<$ *action* $>$ a qual contém a instrução *LogBolt*, que especifica o tipo de ação desejada pelo administrador, que é a geração de um *log* no tempo pré-determinado. Com isso o módulo seleciona o Algoritmo 2.

Algoritmo 2: *Bolt* DE AÇÃO DE DESCOBERTA DE REDE

```
1  ipsMap : IPs de pacotes provenientes da correlacao
2  ipMac : MAC de determinado equipamento
3  arq : arquivo de descoberta de rede
4  begin
5      Evento e disparado quando um sinal s chega :
6      //Inicializacao
7      arq = File("DescRede.txt") ipsMap = StringArray()
8      IniciaIPsDeAuditoria(ipsMap)
9      for ip in ipsMap do
10         ipMac = Arp(ip)
11         Escreve(ip + " : " + RequisitaFabricante(ipMac))
12     end
13 end
```

O Algoritmo 2 irá acessar o arquivo XML gerado pelo Algoritmo 1 no intervalo de tempo descrito na *tag* $<$ $correlationInfo$ $>$, esse arquivo XML é usado com um serviço externo para localização de fabricante do dispositivo a partir do MAC, com isso gerando um arquivo .txt de descoberta de rede com os dados de IP e marca dos fabricantes de cada dispositivos conectado a rede.

5.3 DESCOBERTA DE FALHA EM SERVIDOR *WEB*

Uma das atividades da gerência de falhas é a verificação de disponibilidade de serviços. A fim de validar este estudo de caso, é mantido uma rede de computadores básica com um computador, um servidor *Web*, uma impressora, e outros dispositivos heterogêneos. A partir desses fluxos de rede foram analisados diversos fluxos de dados heterogêneos e com isso posto o protótipo em prática o funcionamento do protótipo no monitoramento de disponibilidade de serviço.

Neste estudo de caso, o objetivo é avaliar o efeito da política aplicada no sistema de correlação de dados em um ambiente *Big Data* com o intuito de monitorar a disponibilidade de um servidor *Web*. A simulação desse estudo de caso também foi realizada tendo como base o cenário, presente na Figura 10. A política utilizada para verificar a disponibilidade de um

servidor *Web* é apresentada na Figura 12.

Na Figura 12 é apresentada a política utilizada para realização de monitoramento de disponibilidade de servidor *web*. Inicialmente o sistema irá executar a nova política de nome *AckTest*. Logo após o módulo *Decision-Making* irá acessar os principais comandos, $<$ *trigger* $>$, $<$ *correlation* $>$, $<$ *correlationType* $>$ e $<$ *action* $>$ para realizar o processamento dos fluxos de dados e a geração de uma ação esperada pelo administrador.

Figura 12 – Política usada para descoberta de falha de seridor web

```
<politicsConf>
        <confID>AckTest</confID>
        <basicInfo>
                <trigger>ChronoBolt</trigger>
                <correlation>AckLogBolt</correlation>
                <correlatType>ProtocolServiceBolt</correlatType>
                <action>LogBolt</action>
        </basicInfo>
        <correlationInfo>
                <checkList>
                        <ipAdress>
                                <source>10.0.10.88</source>
                                <dest>200.132.35.220</dest>
                        </ipAdress>
                        <protocol>TCP</protocol>
                        <service>HTTP</service>
                        <port>80</port>
                        <timeMSAmount>7000</timeMSAmount>
                        <eventsAmount>5</eventsAmount>
                </checkList>
        </correlationInfo>
        <actionInfo>
                <logLocal>/home/storm/StormInfrastructure
                /Storm/apache-storm-1.0.3/examples
                /storm-starter/src/jvm/storm
                /starter/Log</logLocal>
                <email>storm@storm.com.br</email>
        </actionInfo>
</politicsConf>
```

Fonte: acervo pessoal.

Após o acesso aos comando principais, o módulo *Decision-Making* irá traduzir a *tag* $<$ *correlationType* $>$ para um método interno do sistema, especificando o tipo de formatação necessária dos fluxos de dados para posterior processamento destes fluxos. A formatação de fluxos de dados pelo método *ProtocolServiceBolt* modela os fluxos de diferentes ferramentas em um formato idêntico, tornando-os disponíveis para analise do sistema.

Logo após o módulo irá traduzir a *tag* $<$ *trigger* $>$ que irá identificar o gatilho da ação, que nesse caso é por tempo (*i.e,* a cada X milissegundos de processamento será desencadeado uma ação).

Após a definição do modelo de formatação dos fluxos e do gatilho da ação, o módulo interpretará a *tag* $<$ *correlation* $>$ preenchida com o comando *AckLogBolt*, que especifica o tipo de correlação usada, que nesse caso é uma filtragem de dados. O *Decision-Making* irá selecionar o Algoritmo 3 de filtragem de dados, e determinará quais fluxos devem ser analisados, tomando com base o elemento $<$ *correlationInfo* $>$.

Algoritmo 3: *Bolt* DE CORRELAÇÃO PARA DISPONIBILIDADE DE DISPOSITIVO

```
1  ackMap : mapeamento de pacotes confirmados
2  begin
3      //Inicializacao
4      ackMap = HashMap()
5      Evento e disparado quando um pacote p chega :
6      if p.EXISTE(frame_Ack) then
7          if ackMap.CONTEM(p.frameAck) then
8              ackMap.REMOVE(p.frameAck)
9          end
10     end
11     ackMap.INICIA(p.frame)
12     ackMap.INSERE(p.frame, p)
13
14     Evento e disparado quando um sinal de acao s chega :
15     GerarXMLCorrelacao(ackMap)
16     EnviarEventoAcao()
17 end
```

O Algoritmo 3 analisará somente os fluxos de rede que possuírem os campos:

a) $<$ *source* $>$ com endereço ip origem = 10.0.10.88;

b) $<$ *dest* $>$ com endereço ip destino = 200.132.35.220;

c) $<$ *protocol* $>$ protocolo = TCP;

d) $<$ *service* $>$ serviço = HTTP, pois o que está sendo monitorado é um serviço *Web*;

e) $<$ *port* $>$ porta = 80;

f) $<$ *timeMSAmount* $>$ todos os fluxos serão analisados a cada 7000 milissegundos;

g) $<$ *eventsAmount* $>$ não serão analisados por número de fluxos.

Com base nos campos da *tag* $< correlationInfo >$ o Algoritmo 3 irá captar os fluxos de dados formatados do sistema, depois comprimirá os dados, com isso gerando um arquivo XML com uma lista de dados não repetidos a cada intervalo de tempo definido na *tag* $< correlationInfo >$.

O módulo *Decision-Making* interpretará a *tag* $<$ *action* $>$ a qual contém a instrução *LogBolt*, que especifica o tipo de ação desejada pelo administrador, que é a geração de um *log* . Com isso o módulo seleciona o Algoritmo 4.

Algoritmo 4: AÇÃO (ACKLOG)

```
1  unconfirmedFrames : frames que nao receberam ack
2  unconfirmedData : pacotes completos referentes a frame sem ack
3  arq : arquivo de log
4  begin
5      Evento e disparado quando um sinal s chega :
6      //Inicializacao
7      arq = File("AckLog.txt") unconfirmedFrames = StringArray()
8      unconfirmedData = StringHash()
         IniciaDadosPorIP(unconfirmedFrames, unconfirmedData)
9      for frame in unconfirmedFrames do
10         Escreve(frame + " : " + unconfirmedData.RECUPERA(frame)
11     end
12 end
```

Ao final do processamento do algoritmo 3 é gerado um arquivo XML com as possíveis falhas do servidor *Web* e disparado um comando invocando a ação presente no Algoritmo 4. O Algoritmo 4 irá acessar o arquivo XML gerado pelo Algoritmo 3 criará um arquivo .txt de disponibilidade de serviço, acessará os pacotes de dados a partir do *frame* ou pacote que não receberam a confirmação do último pacote e será gerado um *log* com as possíveis quedas do servidor *Web*.

Somado a qualidade de políticas presente no sistema, este também possuí técnicas de correlação de eventos em ambiente *Big Data*, o que permite uma análise inteligente e capaz de extrair *insights* dos fluxos de rede analisados. Tudo isso acrescentado a técnica de *Big Data* que pode realizar o processamento dos dados em tempo de execução. Com isso demonstrando que o uso dessas três técnicas podem facilitar o gerenciamento de redes, dessa forma liberando o administrador de rede para outras atividades.

5.4 AVALIAÇÃO

O monitoramento de redes de computadores mostrou-se uma área do conhecimento que requer árduo trabalho e dedicação, tanto para visão acadêmica como empresarial. Devido a sua grande granularidade, essa área do conhecimento é composta de modelos de referência diferentes, vários protocolos, inúmeras maneiras de se monitorar um dado dispositivo, vários serviços e *softwares* que podem auxiliar o administrador de redes.

Através dos estudos de caso foi possível constatar que o uso de politicas é uma abordagem factível de se implementar em ambientes de rede, principalmente para se fazer correlação de dados em ambiente *Big Data*. Foi possível observar vantagens no uso de PBNM, pois o fato de definir as políticas a um nível de abstração elevado cria vantagens, quer no nível técnico ao simplificar o tratamento e administração das estruturas, quer ao nível funcional, ao aproximar a utilização da estrutura dos objetivos pretendidos, além de constatar que ao usar correlação de dados em ambiente *Big Data* não houve uma sobrecarga na rede com mensagens de requisição-resposta. Com o aumento do número de fluxos de rede percebeu-se que o sistema comportou-se escalável pois foi implementado em ambiente Apache Storm, o qual faz paralelismo de processos.

O uso de técnicas de correlação de dados em ambiente *Big Data* no monitoramento de redes, já é usado por outros pesquisadores com o intuito de simplificação de mensagens, além de tratar dados heterogêneos e de possuir a capacidade de ser usado em monitoramento de redes em tempo continuo.

Já o uso de PBNM surgiu da necessidade de controle de QoS e de configurações de rede em geral. Mesmo havendo vantagens de integração, simplificação e homogenização de configuração e analise dos dados por todo um domínio de controle, não houve uma expansão do uso para outras áreas (*i.e,* no monitoramento de redes em geral), faltando uma normalização necessária que satisfaça a maioria das comunidades interessadas nessa técnica.

Com os estudos de caso foi possível perceber que há uma limitação do protótipo quando o administrador, ao preencher a política em XML, deve colocar os campos (`<trigger>`, `<correlation>`, `<correlatType>` e `<action>`) iguais aos nomes colocados em suas descrições em seus respectivos algoritmos. No caso de haver uma diferença de nomes no preenchimento do campo, o serviço não será executado corretamente e a solução para este problema vai além do escopo deste trabalho.

6 CONSIDERAÇÕES FINAIS

As redes de computadores estão em permanente crescimento. A integração de inúmeras tecnologias, o progressivo aumento no número de usuários, acabam tornando o monitoramento de redes uma tarefa bastante complexa e que demanda conhecimentos do sistema a ser monitorado, pois há uma crescente heterogeneidade e complexidade dos novos dispositivos adicionados.

Nesta dissertação foi proposto uma arquitetura de monitoramento de redes, baseada em políticas que utiliza correlação de dados em ambiente *Big Data*, a qual possibilita a integração de diferentes algoritmos de correlação, usados para analise de fluxos, com algoritmos de ações. Esta arquitetura permite que os administradores interessados no monitoramento de determinados dispositivos ou serviços de rede, possam desenvolver uma política e registrá-la no sistema a fim de realizar monitoramento dos mesmos. Além desta arquitetura, foram revistos alguns conceitos de monitoramento de redes, PBNM, correlação de dados e *Big Data*. Por fim, foi desenvolvido um modelo de política em XML a ser usado pelo administrador no sistema proposto.

Um protótipo baseado na arquitetura de monitoramento proposta foi desenvolvido com o objetivo de validar os módulos definidos. Com esse objetivo foi desenvolvido dois estudos de caso, com os quais foi possível perceber que ao usar políticas no monitoramento de redes existem algumas vantagens, pois ao definir políticas a um nível de abstração elevado fez com que as estruturas utilizadas fossem aproximadas dos objetivos almejados. Também foi possível constatar que não houve um número elevado de troca de mensagens com a finalidade de monitoramento de rede. Com isso demonstrando uma redução na complexidade e consequentes ganhos de facilidade e velocidade do monitoramento de redes.

Finalmente, com base nas conclusões obtidas com os resultados obtidos através da validação do protótipo desenvolvido, pode-se concluir que, de fato, monitoramento de redes baseado em política utilizando correlação de dados em um ambiente *Big Data* pode ser utilizado como mecanismo de monitoramento de redes.

Como trabalhos futuros pretende-se integrar nos algoritmos de correlação (*e.g.*, Raciocínio Baseado em Casos e Correlação Distribuída Baseada em Políticas), além de algoritmos de ação (*e.g.*, *script* de correção) à estrutura da arquitetura proposta. Com estes novos algoritmos de correlação e ação, outros experimentos deverão ser realizados, envolvendo novos

cenários. Também pretende-se criar um interface gráfica para cadastro de novas políticas de monitoramento de redes. Ainda que a implementação efetuada nesta dissertação seja apenas um protótipo, pode-se afirmar que a mesma está operacional e pode ser prontamente usada para monitoramento de redes.

REFERÊNCIAS

AHMED, T.; MEHAOUA, A.; BOUTABA, R. Dynamic QoS adaptation using COPS and network monitoring feedback. In: IFIP/IEEE INTERNATIONAL CONFERENCE ON MANAGEMENT OF MULTIMEDIA NETWORKS AND SERVICES. **Anais...** [S.l.: s.n.], 2002. p.250–262.

BATISTA, B. L. A.; FERNANDEZ, M. P. Ponderflow: a new policy specification language to sdn openflow-based networks. **International Journal on Advances in Networks and Services**, [S.l.], v.7, n.3 & 4, 2014.

BELLER, A. Uma Arquitetura para Gerenciamento de QoS Baseado em Políticas. **Pontifícia Universidade Católica do Paraná**, [S.l.], 2005.

BRAY, T. et al. **Extensible markup language (XML) 1.0**. [S.l.]: W3C recommendation, 2008.

BRETERNITZ, V. J.; SILVA, L. A. Big data: um novo conceito gerando oportunidades e desafios. **Revista Eletrônica de Tecnologia e Cultura**, [S.l.], v.2, n.2, 2013.

CASADO, M. et al. Fabric: a retrospective on evolving sdn. In: HOT TOPICS IN SOFTWARE DEFINED NETWORKS. **Proceedings...** [S.l.: s.n.], 2012. p.85–90.

CHENG, S. et al. NEPnet: a scalable monitoring system for anomaly detection of network service. In: INTERNATIONAL CONFERENCE ON NETWORK AND SERVICES MANAGEMENT, 7. **Proceedings...** [S.l.: s.n.], 2011. p.338–342.

COSTA, J. S. da et al. Um módulo de negociação baseado em políticas para a gerência de grades computacionais. In: III WORKSHOP ON GRID COMPUTING AND APPLICATIONS III WORKSHOP DE COMPUTAÇÃO EM GRID E APLICAÇÕES. **Anais...** [S.l.: s.n.], 2005. p.92.

DAMIANOU, N. et al. The ponder policy specification language. In: **Policies for Distributed Systems and Networks**. [S.l.]: Springer, 2001. p.18–38.

FACHINELLI, A. C. Big Data: o novo desafio para gestão. **Revista Inteligência Competitiva**, [S.l.], v.4, n.1, p.18–38, 2014.

GOPALKRISHNAN, V. et al. Big data, big business: bridging the gap. In: INTERNATIONAL WORKSHOP ON BIG DATA, STREAMS AND HETEROGENEOUS SOURCE MINING: ALGORITHMS, SYSTEMS, PROGRAMMING MODELS AND APPLICATIONS, 1. **Proceedings...** [S.l.: s.n.], 2012. p.7–11.

GRANVILLE, L. Z. et al. A pbnm system for integrated qos and multicast management. In: POLICIES FOR DISTRIBUTED SYSTEMS AND NETWORKS, 2003. PROCEEDINGS. POLICY 2003. IEEE 4TH INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2003. p.243–246.

JAKOBSON, G.; WEISSMAN, M. Alarm correlation. **IEEE network**, [S.l.], v.7, n.6, p.52–59, 1993.

JUNIOR, J. C. d. S. F. et al. Big data e gestão do conhecimento: definições e direcionamentos de pesquisa. **Revista Alcance (Online)**, [S.l.], v.23, n.4, p.529, 2016.

KRYFTIS, Y. et al. Policy-Based Management for Federation of Virtualized Infrastructures. **Journal of Network and Systems Management**, [S.l.], v.25, n.2, p.229–252, 2017.

LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. D. **Mining of massive datasets**. [S.l.]: Cambridge university press Cambridge, 2014.

LUCKHAM, D.; SCHULTE, R. **Event processing glossary—version 1.1. Event Processing Technical Society, July 2008**. 2008.

MANYIKA, J. et al. Big data: the next frontier for innovation, competition, and productivity. , [S.l.], 2011.

MATOS, D. R.; PAILLARD, G. A.; CASTRO, M. F. de. **A Distributed Coverage Node Scheduling Algorithm for Dense Wireless Sensor Networks.** , [S.l.], 2016.

MULAHUWAISH, A. A.; BAKAR, K. A.; GHAFOOR, K. Z. A Congestion Avoidance Approach in Jumbo Frame-enabled IP Network. **Editorial Preface**, [S.l.], v.3, n.1, 2012.

OLIVEIRA, H. T. et al. **Estudo sobre caracterısticas de administradores de redes de computadores no Brasil para identificaç ao e elaboraç ao de personas.** , [S.l.], 2017.

RIBEIRO, M. B. **Um Sistema para monitoração de redes IP baseado em políticas.** , [S.l.], 2003.

ROBITZSCH, S. et al. E-stream: towards pattern centric network incident discovery and corrective action recommendation in telecommunication networks. In: INTEGRATED NETWORK MANAGEMENT (IM), 2015 IFIP/IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2015. p.842–845.

SALAH, S.; MACIÁ-FERNÁNDEZ, G.; DÍAZ-VERDEJO, J. E. A model-based survey of alert correlation techniques. **Computer Networks**, [S.l.], v.57, n.5, p.1289–1317, 2013.

SLOMAN, M. Policy driven management for distributed systems. **Journal of network and Systems Management**, [S.l.], v.2, n.4, p.333–360, 1994.

TOSHNIWAL, A. et al. Storm@ twitter. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2014. **Proceedings...** [S.l.: s.n.], 2014. p.147–156.

ZAMAN, F. et al. i-MagNet: a real-time intelligent framework for finding specific needles from needle stacks. In: INTEGRATED NETWORK MANAGEMENT (IM), 2015 IFIP/IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2015. p.846–849.

ZHOU, E. et al. **Network management system**. US Patent 9,680,713.

APÊNDICES

APÊNDICE A – *Bolt* de descoberta de rede

Neste apêndice é apresentado o *Bolt* de descoberta de rede, cujo o código presente na *Listing* A.1, o qual é responsavel por realizar as correlações de eventos presentes no fluxos de redes capturados e injetados na arquitetura e com isso fazer uma verificação de rede descobrindo os disposivo ativos na rede.

Listing A.1 – *Bolt* de descoberta de rede

```
1
package storm.starter.ActionBase;
3
import java.util.Map;
5 import java.util.List;
import java.util.ArrayList;
7 import java.io.File;
import java.io.FileWriter;
9 import java.io.IOException;

11 import storm.starter.AlgorithmBase.PoliticsXML;
import storm.starter.AlgorithmBase.CorrelationXML;
13 import storm.starter.AlgorithmBase.AuditElement;

15 import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
17 import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
19 import org.apache.storm.topology.IRichBolt;
import org.apache.storm.topology.OutputFieldsDeclarer;
21 import org.apache.storm.tuple.Tuple;
import org.apache.commons.io.FileUtils;
23
public class AuditLogBolt implements IRichBolt {
25 private String logOutPath;
private String basePath, politicsPath, politicsName;
27 private PoliticsXML configuration;
private OutputCollector collector;
29 private Integer logNumber;

31 public AuditLogBolt(String basePath, String politicsName){
this.politicsPath = basePath + politicsName;
33 this.basePath = basePath;
this.politicsName = politicsName;
```

```
35 this.logNumber = 1;
   }
37
   @Override
39 public void prepare(Map conf, TopologyContext context,+
                        +OutputCollector collector) {
41 this.configuration = new PoliticsXML(this.politicsPath);
   this.logOutPath = this.configuration.getAILogLocal();
43 this.collector = collector;
   }
45
   @Override
47 public void execute(Tuple tuple) {
   try{
49 ArrayList<String> consideredIPs = new ArrayList<String> ();
   ArrayList<AuditElement> auditedIPs = new ArrayList<AuditElement> ();
51 CorrelationXML request = new CorrelationXML(tuple.getString(0));
   FileWriter finserter = new FileWriter(new File(this.logOutPath +
53                              + this.logNumber.toString() + ".txt"));
   this.logNumber++;
55 finserter.write("============= DISCOVERY LOG ============\n\n");
   while (request.chargeNextElement()){
57 String[] packets = request.getElementPackets().split("\n");
   for(String packet : packets){
59 String[] packetData = packet.split(",");
   if (!consideredIPs.contains(packetData[0])){
61 consideredIPs.add(packetData[0]);
   auditedIPs.add(new AuditElement(packetData[0]));
63 }
   if (!consideredIPs.contains(packetData[1])){
65 consideredIPs.add(packetData[1]);
   auditedIPs.add(new AuditElement(packetData[1]));
67 }
   }
69 for(AuditElement audited : auditedIPs){
   finserter.write(audited.getIP() + " -> " +
71             + audited.getVendor() + "\n");
   }
73 finserter.write("\n");
   }
75 finserter.close();
   request.close();
77 }
   catch(IOException ex){}
79
   collector.ack(tuple);
81 }
```

```
83  @Override
    public void cleanup() {
85  try{
    File dir = new File(this.basePath + this.politicsName + "TMP");
87  FileUtils.deleteDirectory(dir);
    }
89  catch(IOException ex){}

91  }

93  @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {}
95
    @Override
97  public Map<String, Object> getComponentConfiguration() {
    return null;
99  }
    }
```

APÊNDICE B – *Bolt* de descoberta de falha em servidor web

Neste apêndice é apresentado o *Bolt* de disponibilidade de serviço, cujo o código presente na *Listing* B.1, o qual é responsavel por realizar as correlações de eventos presentes no fluxos de redes capturados e injetados na arquitetura e com isso verificar se houve queda no servidor web monitorado.

Listing B.1 – *Bolt* de disponibilidade de serviço

```
2  package storm.starter.ActionBase;

4  import java.util.Map;
   import java.util.List;
6  import java.util.ArrayList;
   import java.io.File;
8  import java.io.FileWriter;
   import java.io.IOException;
10
   import storm.starter.AlgorithmBase.PoliticsXML;
12 import storm.starter.AlgorithmBase.CorrelationXML;

14 import org.apache.storm.tuple.Fields;
   import org.apache.storm.tuple.Values;
16 import org.apache.storm.task.OutputCollector;
   import org.apache.storm.task.TopologyContext;
18 import org.apache.storm.topology.IRichBolt;
   import org.apache.storm.topology.OutputFieldsDeclarer;
20 import org.apache.storm.tuple.Tuple;
   import org.apache.commons.io.FileUtils;
22
   public class AckLogBolt implements IRichBolt {
24     private String logOutPath;
       private String basePath, politicsPath, politicsName;
26     private PoliticsXML configuration;
       private OutputCollector collector;
28     private Integer logNumber;

30     public AckLogBolt(String basePath, String politicsName){
           this.politicsPath = basePath + politicsName;
32         this.basePath = basePath;
           this.politicsName = politicsName;
34         this.logNumber = 1;
```

```
    }
36
    @Override
38  public void prepare(Map conf, TopologyContext context,
                        ,OutputCollector collector) {
40      this.configuration = new PoliticsXML(this.politicsPath);
        this.logOutPath = this.configuration.getAILogLocal();
42      this.collector = collector;
    }
44
    @Override
46  public void execute(Tuple tuple) {
    try{
48  CorrelationXML request = new CorrelationXML(
                                              tuple.getString(0));
50  FileWriter finserter = new FileWriter(new File(this.logOutPath+
                                  this.logNumber.toString() + ".txt"));
52  this.logNumber++;
    finserter.write(""+
54  "============= UNCONFIRMED MESSAGES =============\n\n");
    while (request.chargeNextElement()){
56  finserter.write("Frame number: " + request.getElementPort() + "\n")
    String[] packets = request.getElementPackets().split("\n");
58  for(String packet : packets){
        finserter.write("Frame details: " + packet + "\n");
60  }
    finserter.write("\n");
62  }
    finserter.close();
64  request.close();
    }
66  catch(IOException ex){}

68  collector.ack(tuple);
}
70
    @Override
72  public void cleanup() {
    try{
74  File dir = new File(this.basePath + this.politicsName + "TMP");
    FileUtils.deleteDirectory(dir);
76  }
    catch(IOException ex){}
78
}
80
    @Override
82  public void declareOutputFields(OutputFieldsDeclarer declarer) {}
```

```
84      @Override
        public Map<String, Object> getComponentConfiguration() {
86          return null;
        }
88  }
```

APÊNDICE C – *Bolt* de ação para *log* disponibilidade de serviço

Neste apêndice é apresentado o *Bolt* de ação para *log* disponibilidade de serviço, cujo o código presente na *Listing* C.1, o qual é responsavel por realizar as ações que o administrador estipulou para serem realizadas ao final do processamento do fluxos de rede injetados no sistema.

Listing C.1 –
textitBolt de ação para *log* disponibilidade de serviço

```
2  package storm.starter.ActionBase;

4  import java.util.Map;
   import java.util.List;
6  import java.util.ArrayList;
   import java.io.File;
8  import java.io.FileWriter;
   import java.io.IOException;
10
   import storm.starter.AlgorithmBase.PoliticsXML;
12 import storm.starter.AlgorithmBase.CorrelationXML;

14 import org.apache.storm.tuple.Fields;
   import org.apache.storm.tuple.Values;
16 import org.apache.storm.task.OutputCollector;
   import org.apache.storm.task.TopologyContext;
18 import org.apache.storm.topology.IRichBolt;
   import org.apache.storm.topology.OutputFieldsDeclarer;
20 import org.apache.storm.tuple.Tuple;
   import org.apache.commons.io.FileUtils;
22
   public class AckLogBolt implements IRichBolt {
24    private String logOutPath;
      private String basePath, politicsPath, politicsName;
26    private PoliticsXML configuration;
      private OutputCollector collector;
28    private Integer logNumber;

30    public AckLogBolt(String basePath, String politicsName){
         this.politicsPath = basePath + politicsName;
32       this.basePath = basePath;
         this.politicsName = politicsName;
```

```
34          this.logNumber = 1;
        }
36
        @Override
38      public void prepare(Map conf, TopologyContext context, OutputCollect
           this.configuration = new PoliticsXML(this.politicsPath);
40         this.logOutPath = this.configuration.getAILogLocal();
           this.collector = collector;
42      }

44      @Override
        public void execute(Tuple tuple) {
46        try{
             CorrelationXML request = new CorrelationXML(tuple.getString(0))
48           FileWriter finserter = new FileWriter(new File(this.logOutPath
             this.logNumber++;
50           finserter.write("============= UNCONFIRMED MESSAGES ==========
         while (request.chargeNextElement()){
52              finserter.write("Frame number: " + request.getElementPort()
                String[] packets = request.getElementPackets().split("\n");
54              for(String packet : packets){
                   finserter.write("Frame details: " + packet + "\n");
56              }
                finserter.write("\n");
58           }
             finserter.close();
60           request.close();
          }
62        catch(IOException ex){}

64        collector.ack(tuple);
        }
66
        @Override
68      public void cleanup() {
          try{
70            File dir = new File(this.basePath + this.politicsName + "TMP")
                  FileUtils.deleteDirectory(dir);
72        }
          catch(IOException ex){}
74


76      }

78      @Override
        public void declareOutputFields(OutputFieldsDeclarer declarer) {}
80
        @Override
```

```
82      public Map<String, Object> getComponentConfiguration() {
            return null;
84      }
    }
```

APÊNDICE D – *Bolt* de ação para *log* de descoberta de rede

Neste apêndice é apresentado o *Bolt* de ação para *log* descoberta de rede, cujo o código presente na *Listing* D.1, o qual é responsavel por gerar um *log* dos dispositivos descobertos após o final do processamento do fluxos de rede injetados no sistema.

Listing D.1 –
textitBolt de ação para *log* de descoberta de rede

```
2  package storm.starter.ActionBase;

4  import java.util.Map;
   import java.util.List;
6  import java.util.ArrayList;
   import java.io.File;
8  import java.io.FileWriter;
   import java.io.IOException;
10
   import storm.starter.AlgorithmBase.PoliticsXML;
12 import storm.starter.AlgorithmBase.CorrelationXML;
   import storm.starter.AlgorithmBase.AuditElement;
14
   import org.apache.storm.tuple.Fields;
16 import org.apache.storm.tuple.Values;
   import org.apache.storm.task.OutputCollector;
18 import org.apache.storm.task.TopologyContext;
   import org.apache.storm.topology.IRichBolt;
20 import org.apache.storm.topology.OutputFieldsDeclarer;
   import org.apache.storm.tuple.Tuple;
22 import org.apache.commons.io.FileUtils;

24 public class AuditLogBolt implements IRichBolt {
      private String logOutPath;
26    private String basePath, politicsPath, politicsName;
      private PoliticsXML configuration;
28    private OutputCollector collector;
      private Integer logNumber;
30
      public AuditLogBolt(String basePath, String politicsName){
32       this.politicsPath = basePath + politicsName;
         this.basePath = basePath;
34       this.politicsName = politicsName;
```

```
        this.logNumber = 1;
36    }

38    @Override
      public void prepare(Map conf, TopologyContext context, OutputCollect
40        this.configuration = new PoliticsXML(this.politicsPath);
          this.logOutPath = this.configuration.getAILogLocal();
42        this.collector = collector;
      }
44
      @Override
46    public void execute(Tuple tuple) {
        try{
48     ArrayList<String> consideredIPs = new ArrayList<String> ();
       ArrayList<AuditElement> auditedIPs = new ArrayList<AuditElement> ()
50         CorrelationXML request = new CorrelationXML(tuple.getString(0))
           FileWriter finserter = new FileWriter(new File(this.logOutPath
52         this.logNumber++;
           finserter.write("============= AUDIT LOG =============\n\n");
54     while (request.chargeNextElement()){
              String[] packets = request.getElementPackets().split("\n");
56            for(String packet : packets){
             String[] packetData = packet.split(",");
58           if (!consideredIPs.contains(packetData[0])){
           consideredIPs.add(packetData[0]);
60         auditedIPs.add(new AuditElement(packetData[0]));
             }
62           if (!consideredIPs.contains(packetData[1])){
                   consideredIPs.add(packetData[1]);
64                 auditedIPs.add(new AuditElement(packetData[1]));
                 }
66            }
         for(AuditElement audited : auditedIPs){
68        finserter.write(audited.getIP() + " -> " + audited.getVendor()
         }
70           finserter.write("\n");
          }
72        finserter.close();
          request.close();
74      }
        catch(IOException ex){}
76
        collector.ack(tuple);
78    }

80    @Override
      public void cleanup() {
82      try{
```

```
            File dir = new File(this.basePath + this.politicsName + "TMP")
84              FileUtils.deleteDirectory(dir);
        }
86      catch(IOException ex){}

88

    }
90

    @Override
92  public void declareOutputFields(OutputFieldsDeclarer declarer) {}

94  @Override
    public Map<String, Object> getComponentConfiguration() {
96      return null;
    }
98  }
```