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

Vinícius Teixeira

UM SISTEMA PARA PROCESSAMENTO DE DADOS EM TEMPO REAL APLICADO À ANÁLISE E MONITORAMENTO DE REDE

Santa Maria, RS
2019

Vinícius Teixeira

UM SISTEMA PARA PROCESSAMENTO DE DADOS EM TEMPO REAL APLICADO À ANÁLISE E MONITORAMENTO DE REDE

Trabalho Final de Graduação apresentado ao Programa de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

ORIENTADOR: Prof. João Vicente Ferreira Lima

462
Santa Maria, RS
2019

Vinícius Teixeira

UM SISTEMA PARA PROCESSAMENTO DE DADOS EM TEMPO REAL APLICADO À ANÁLISE E MONITORAMENTO DE REDE

Trabalho Final de Graduação apresentado ao Programa de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

Aprovado em 3 de dezembro de 2019:

João Vicente Ferreira Lima, Dr. (UFSM)
(Orientador)

Patrícia Pitthan de Araújo Barcelos, Dr. (UFSM)

Celio Trois, Dr. (UFSM)

Santa Maria, RS
2019

RESUMO

UM SISTEMA PARA PROCESSAMENTO DE DADOS EM TEMPO REAL APLICADO À ANÁLISE E MONITORAMENTO DE REDE

AUTOR: Vinícius Teixeira
ORIENTADOR: João Vicente Ferreira Lima

Com o crescimento exponencial do número de dispositivos conectados à rede, cada vez mais dados são gerados e transmitidos de forma contínua pela rede. Neste contexto, o monitoramento apropriado das redes de computadores se torna cada vez mais primordial. A detecção tardia de anomalias muitas vezes provoca o aumento substancial dos riscos de danos irreparáveis e inviabiliza tentativa de defesa, o que faz com que seja crucial obter informações atualizadas e em tempo real sobre o fluxo de dados gerado pelo tráfego de rede. Nesse contexto, arquiteturas de processamento de *Big Data* em tempo real, como Lambda e Kappa, vêm tendo destaque nos últimos anos. O presente trabalho tem como objetivo o desenvolvimento de um sistema baseado na arquitetura Lambda, aplicado ao monitoramento e processamento do fluxo de dados do tráfego de rede, integrando diferentes ferramentas de código aberto para realizar desde a coleta dos dados até o armazenamento dos dados processados. A análise experimental do sistema ocorre simulando mais de 320.000 conexões súbitas na rede local monitorada, às quais o sistema levou em média 1 minuto e 36 segundos para processar e salvar os resultados no banco de dados.

Palavras-chave: Monitoramento de rede. *Big Data*. Tempo real.

ABSTRACT

A SYSTEM FOR REAL TIME DATA PROCESSING APPLIED TO NETWORK ANALYSIS AND MONITORING

AUTHOR: Vinícius Teixeira
ADVISOR: João Vicente Ferreira Lima

As the number of devices connected to the network grows exponentially, more and more data is generated and transmitted in the form of continuous streams over the network. In this context, proper monitoring of computer networks becomes increasingly paramount. Late detection of anomalies often causes or substantially increases the risk of irreparable damage and makes a defense attempt unfeasible, what makes crucial to obtain up-to-date real-time information about the flow of data generated by network traffic. In this context, Big Data architectures for real time processing, such as Lambda and Kappa, have been highlighted in recent years. The present work aims to develop a system based on Lambda architecture, applied to the monitoring and processing of network traffic data flow, integrating different open source tools to perform from data collection to data storage. An experimental analysis of the system takes place simulating over 320,000 sudden connections on the monitored local network, which the system took an average of 1 minute and 36 seconds to process and save the results on the database.

Keywords: Network monitoring. Big Data. Real time

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

De acordo com (MARZ; WARREN, 2015), mais de 30.000 gigabytes de dados são gerados a cada segundo. Isto se deve principalmente à rápida evolução da Internet, onde o volume, velocidade e variedade de dados gerados e transmitidos pela rede vem crescendo cada vez mais, exigindo muitas vezes uma infraestrutura adequada para realizar o monitoramento e processamento destas informações.

Neste contexto, surge o conceito de *Big Data*, para lidar com ingestão, processamento e análise de dados grandes ou complexos demais para sistemas de banco de dados tradicionais(DUMBILL, 2012). Os sistemas implementados sobre este conceito são utilizados para realizar análise de grandes conjuntos de dados com o intuito de obter informações relevantes sobre os mesmos, aplicando diferentes métodos de processamento. Uma área de aplicação destes sistemas é para efetuar o monitoramento do tráfego de rede, a fim de obter desde informações sobre o consumo de banda da rede a identificar anomalias que ocorrem.

Com o intuito de fazer essa análise, são frequentemente implementados meios que realizam o processamento de dados em lote, que fornece bons resultados sobre o que ocorreu no passado (SOUMAYA et al., 2017), contudo não considera o que acontece em tempo real, sendo impróprio para disponibilizar resultados com baixa latência. Com o objetivo de apresentar soluções a esse problema, novas arquiteturas foram propostas, uma delas é a arquitetura Lambda, abordada por (MARZ, 2011). Esta arquitetura unifica o processamento em lote com o processamento em tempo real, e o resultado de ambos processamentos são exibidos, disponibilizando assim, baixa latência e resultados acurados (OUNACER et al., 2017).

Um sistema baseado na arquitetura Lambda, capaz de processar e analisar o fluxo de dados do tráfego de rede, foi desenvolvido por (HAAS et al., 2019). Apesar do sistema em questão ser baseado na arquitetura Lambda, foi implementado nele somente o processamento de dados em lote, não contendo o processamento em tempo real, o que acarreta em resultados não atualizados. O presente trabalho tem como objetivo desenvolver um sistema, também baseado na arquitetura Lambda, capaz de processar o fluxo de dados do tráfego de rede em tempo real, disponibilizando informações para o monitoramento da rede, como o número de conexões utilizando cada protocolo, a quantidade de conexões por IP originador e IP respondedor, dentre outras, e complementando o sistema de (HAAS et al., 2019).

2 PROCESSAMENTO DO FLUXO DE DADOS DA REDE

Neste capítulo é realizada inicialmente uma abordagem sobre monitoramento de rede na Seção 2.1. Na sequência, a Seção 2.2 destaca as características e conceitos de *Big Data*, a Seção 2.3 aborda arquiteturas de processamento que são utilizadas na implementação de sistemas de *Big Data* para análise de dados, bem como as características apresentadas por cada uma. A última Seção 2.4 enumera os trabalhos relacionados na área de análise do fluxo de dados da rede.

2.1 MONITORAMENTO DE REDE

Para manter a estabilidade, a confiabilidade e a segurança das redes de computadores, é fundamental monitorar o tráfego a fim de obter desde informações sobre o consumo de banda da rede a identificar anomalias que ocorrem (LOPEZ, 2018). As anomalias de rede são definidas como situações onde os níveis de tráfego apresentam um desvio de seu comportamento normal (ZARPELÃO et al., 2010). Elas podem surgir de diferentes situações, como falhas em elementos da rede, erros de configurações, atividades de *softwares* maliciosos, intrusões cibernéticas, e etc (SILVA, 2016). A identificação de anomalias em redes deve ser preferencialmente realizada em tempo real, para que contramedidas possam ser aplicadas, diminuindo assim os possíveis impactos (LOPEZ et al., 2018).

No monitoramento de rede, dados chegam na forma de *streams*, de diferentes fontes (BÄR et al., 2014). Um dos problemas desse tipo de aplicação é a grande quantidade de dados gerados. Mesmo redes de tamanho moderado geram enormes quantidades de dados. Por exemplo, monitorando um único link Ethernet em execução com utilização de 50% gera um terabyte de dados em algumas horas (*Big Data*) (CLAY, 2015).

2.2 BIG DATA

Vivemos em um mundo cada vez mais interconectado, onde informação e dados são gerados por diversas fontes, como redes sociais, *search engines*, *logs*, e-mails, redes de sensores e outras. Neste contexto, surgiu o termo “big data” que é frequentemente utilizado na indústria, na ciência e em diversos outros campos (OUNACER et al., 2017). *Big Data* é um conceito amplo que normalmente refere-se a

conjuntos de dados que as ferramentas tradicionais, como bancos de dados relacionais, são incapazes de processar e gerenciar em um prazo aceitável, devido ao alto volume e complexidade dos dados, que são gerados a todo instante (SOUMAYA et al., 2017).

Não há, no entanto, uma definição acurada para o conceito, segundo (DUMBILL, 2012), os dados são grandes demais, se movem rápido demais ou não se encaixam nas estruturas das arquiteturas de banco de dados existentes. Para obter valor com esses dados, deve haver uma maneira alternativa de processá-los. (CARTER, 2011) define tecnologias de *Big Data* como uma nova geração de tecnologias e arquiteturas projetadas para extrair valor econômico de volumes muito grandes de uma ampla variedade de dados, permitindo captura, descoberta e/ou análise em alta velocidade.

Big Data é comumente caracterizado por “3 Vs”, referenciando volume, variedade e velocidade. Todavia, muitos autores, como (STOREY; SONG, 2017), consideram que veracidade e valor também são importantes, resultando no "5 Vs".

- **Volume:** Quantidade muito grande de dados coletados com tamanhos de terabytes a zetta bytes e além.
- **Variedade**: Variados tipos de dados, incluindo dados estruturados, não estruturados ou semiestruturados, como banco de dados textual, dados de *streaming*, dados de sensores, imagens, áudios, vídeos, arquivos de *log* e outros.
- **Velocidade**: Velocidade do processamento, análise e visualização dos dados.
- **Veracidade**: Confiabilidade dos dados; o usuário deve ter confiança nos dados para que possa utilizá-los como embasamento para tomar uma decisão.
- **Valor**: Os sistemas não só devem ser projetados para processar um enorme volume de dados de forma eficiente, mas também serem capazes de filtrar os dados mais valiosos.

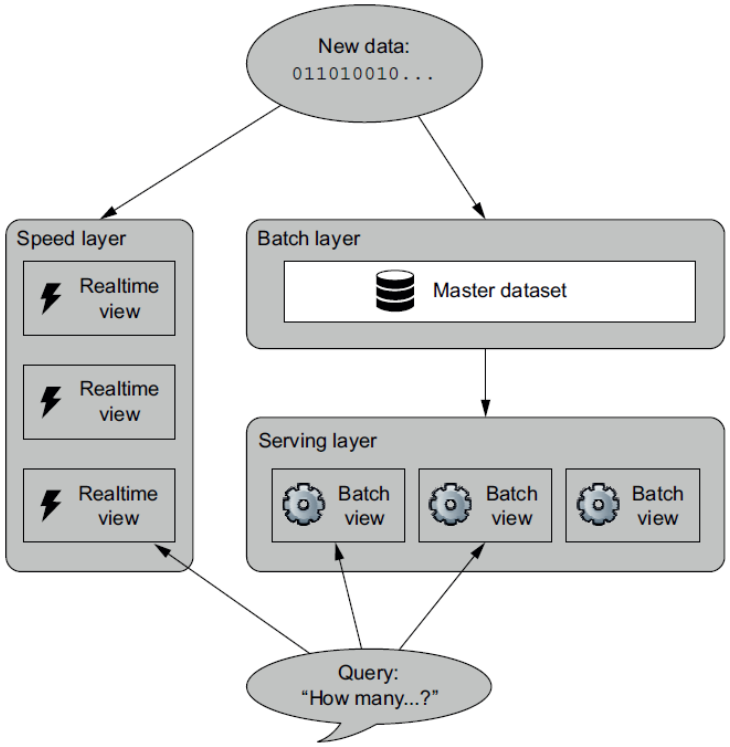
2.3 ARQUITETURAS DE PROCESSAMENTO EM TEMPO REAL

Nesta seção, serão apresentadas duas arquiteturas de processamento de *Big Data* em tempo real, a Lambda e a Kappa, dando destaque à primeira, que é a arquitetura utilizada no presente trabalho.

2.3.1 Arquitetura Lambda

Proposta por (MARZ, 2011), a arquitetura Lambda combina os benefícios do processamento em tempo real e em lote, disponibilizando baixa latência e resultados melhores (SOUMAYA et al., 2017). A ideia principal desta arquitetura é construir sistemas de *Big Data* utilizando um conjunto de camadas, detalhadas a seguir e exibidas Figura 2.1, onde cada camada satisfaz um subconjunto de necessidades.

Figura 2.1 – Diagrama da arquitetura Lambda.



Fonte: (MARZ; WARREN, 2015)

Camada Batch

A camada *Batch* tem duas principais tarefas: armazenar um conjunto de dados mestre imutável em constante crescimento e aplicar arbitrariamente funções nesse conjunto de dados. Este tipo de processamento é feito utilizando sistemas de processamento em lotes.

As funções de processamento aplicadas pela camada *Batch* resultam em *batch views*. A *Batch* executa em um laço *"while(true)"* e reprocessa continuamente todos os dados, gerando a cada processamento novas *views*. As mesmas são armazenadas na camada *Serving*, que viabiliza a consulta e visualização dos resultados.

Camada Serving

A camada *Serving* é um banco de dados distribuído especializado que recebe *views* da camada *Batch* e torna possível a leitura aleatória sobre os dados do banco. Quando novas *batch views* estão disponíveis, a *Serving* automaticamente faz a troca dos dados para que então os novos resultados estejam disponíveis para consulta. As camadas *Batch* e *Serving* proporcionam consultas arbitrárias em um banco de dados arbitrário, no entanto, as consultas são referentes a dados não completamente atualizados.

Camada Speed

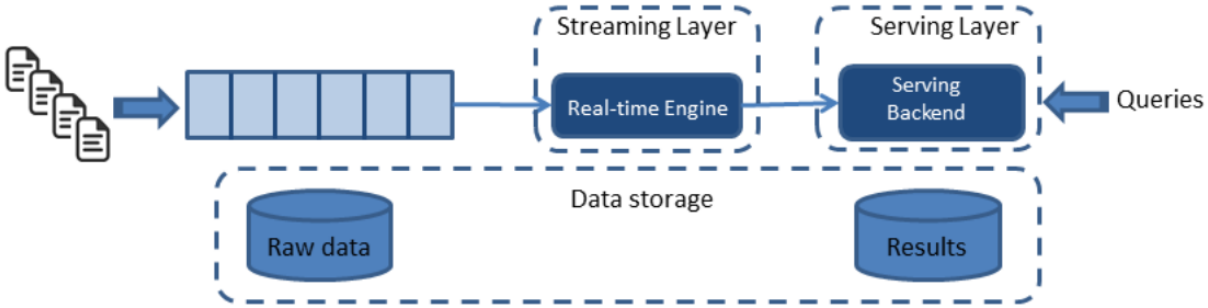
Na camada *Speed* é realizado o processamento dos dados que chegam em tempo real visando baixa latência, armazenando temporariamente os mesmos até que a camada *Batch* termine o processamento em lote. Para alcançar as menores latências possíveis, a camada *Speed* não reprocessa todos os dados como a *Batch*, apenas atualiza as *views* através de incrementos com os dados que são continuamente recebidos.

O propósito da camada *Speed* é compensar o fato de que as *views* da camada *Batch* estão desatualizadas. Incluindo a *Speed* no sistema, os resultados conterão tanto dados recentes quanto os dados processados em lote pela camada *Batch*.

2.3.2 Arquitetura Kappa

A arquitetura Kappa, proposta por (KREPS, 2014), é uma simplificação da arquitetura Lambda, pois mantém o processamento em tempo real e elimina completamente o processamento em lote, como mostra a Figura 2.2. Assim, é voltada para casos que demandem baixa latência sem a necessidade de armazenamento dos dados (LOPEZ et al., 2018). Uma característica desta arquitetura é a relevância da informação mais recente, em detrimento do histórico dos dados (SOBREIRO, 2018). Apesar de possuir mais restrições, na Kappa as *views* são geradas apenas pela camada *Speed*, o que reduz bastante a complexidade do sistema.

Figura 2.2 – Diagrama da arquitetura Kappa.



Fonte: (SEYVET; VIELA, 2016)

2.4 TRABALHOS RELACIONADOS

Existem inúmeros trabalhos na comunidade de pesquisa, que possuem o objetivo de desenvolver soluções para realizar o processamento e análise dos dados provenientes do fluxo de rede. Esta seção apresenta alguns destes trabalhos.

Recentemente, muitos sistemas com o objetivo de realizar processamentos em tempo real têm sido desenvolvidos baseados na arquitetura Lambda. Esta arquitetura foi utilizada no sistema de detecção de ameaças em tempo real proposto por (LOBATO; LOPEZ; DUARTE, 2016), que analisa o tráfego de rede através do processamento de fluxos e com o uso de algoritmos de aprendizado de máquina, classifica ataques e detecta anomalias de maneira automática. Também utilizando a arquitetura Lambda, o sistema abordado por (HAN; NASRIDINOV; RYU, 2018) faz o monitoramento do fluxo de informações geradas por uma mídia social.

Um sistema escalável de medição e análise de tráfego de rede é abordado por (LEE; LEE, 2013), que utiliza a ferramenta Hadoop para efetuar o processamento em lote e análises de IP, TCP, HTTP e NetFlow de vários terabytes de tráfego da Internet. Seguindo nesta linha, um sistema com a finalidade de detectar anomalias na rede é apresentado por (KOZIK, 2017), que faz captura do fluxo de dados de rede e realiza o processamento em lote para posteriormente aplicar aprendizado de máquina para análise das informações obtidas.

O sistema desenvolvido por (HAAS et al., 2019), que serviu de base para o presente trabalho, é baseado na arquitetura Lambda e é capaz de processar e analisar o fluxo de dados do tráfego de rede, com o propósito de identificar anomalias causadas por ataques de *Distributed Denial of Service* (DDoS), força bruta e varredura de portas, sobre os protocolos *Hypertext Transfer Protocol* (HTTP), *Secure Shell* (SSH) e *Transmission Control Protocol* (TCP) respectivamente, e a partir dos resultados, obter informações de organização e país sobre o *Internet Protocol* (IP), responsáveis por estas práticas. Apesar do sistema em questão ser baseado na arquitetura Lambda, o processamento de dados em tempo real não foi implementado, pois na camada *Speed*

é feita apenas a normalização dos dados para efetuar o armazenamento dos mesmos na base de dados da camada *Batch*, sendo assim processados somente em lote, onde aplica-se o algoritmo *K-means* para detecção de anomalias.

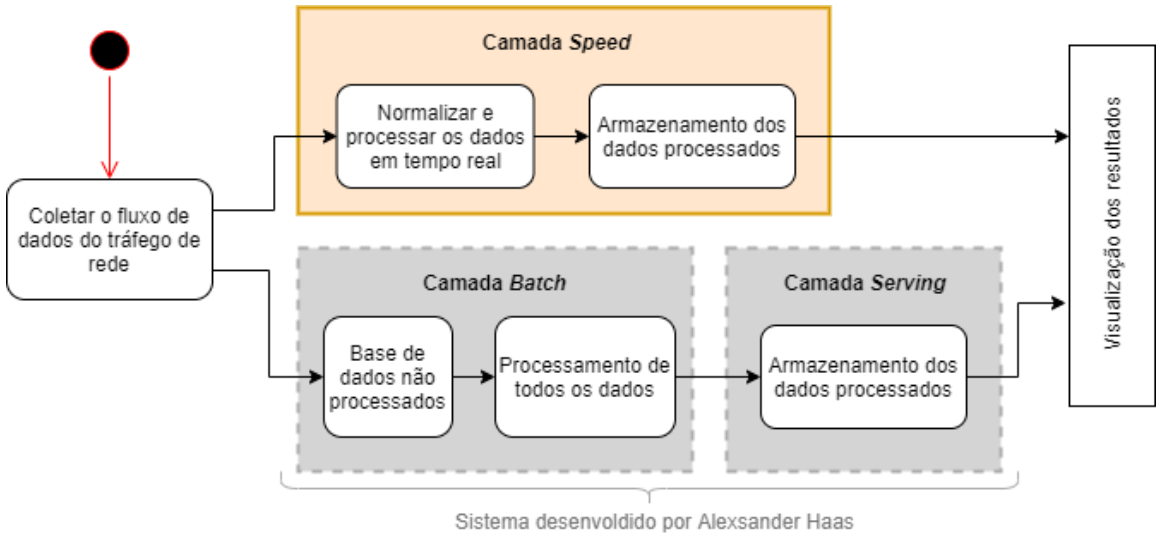
3 DESENVOLVIMENTO

Este capítulo aborda, na Seção 3.1, a arquitetura e as ferramentas utilizadas no desenvolvimento do sistema, e descreve como foi realizado o processamento dos dados na Seção 3.2.

3.1 ARQUITETURA DO SISTEMA

A implementação utilizou a arquitetura Lambda como modelo, focando exclusivamente na camada *Speed*, que é responsável pelo processamento dos dados em tempo real. A arquitetura pode ser separada em algumas etapas que são agrupadas em suas respectivas camadas, conforme pode ser observado na Figura 3.1.

Figura 3.1 – A arquitetura Lambda aplicada ao sistema proposto

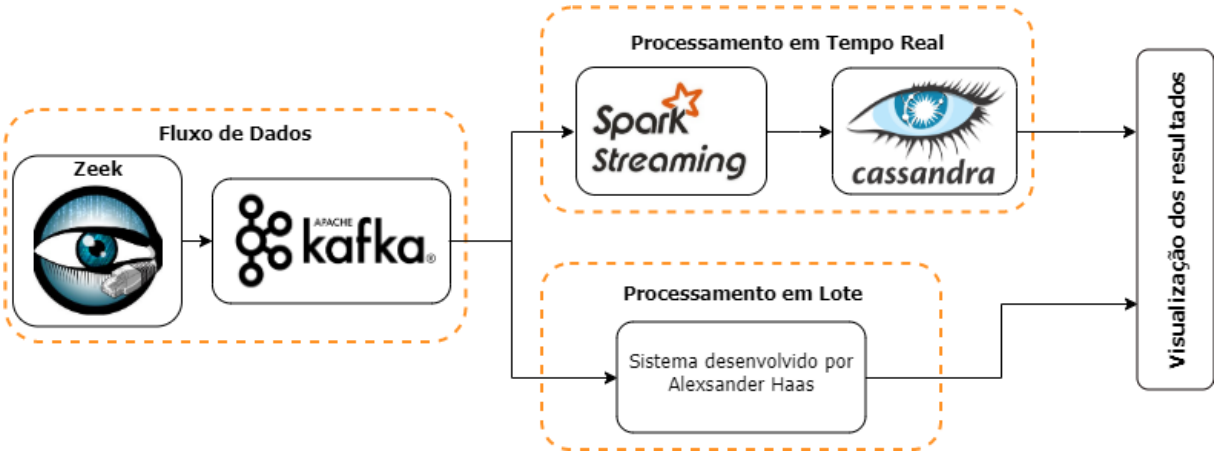


Fonte: O autor

No sistema desenvolvido a primeira etapa é encarregada de coletar o fluxo de dados do tráfego de rede e, então, realizar o transporte das informações para serem processadas. Em seguida, os dados são filtrados de acordo com a característica de sua conexão, fazendo com que os mesmos estejam aptos a serem processados. Depois, é feito o processamento dos dados e as informações são armazenadas incrementalmente no banco de dados. Finalmente, o sistema disponibiliza a visualização dos resultados através de consultas neste banco.

Para a implementação do sistema foram selecionadas diferentes ferramentas de código aberto, onde cada uma atende uma funcionalidade específica e realiza a integração com as demais, conforme exibido na Figura 3.2 e detalhado a seguir.

Figura 3.2 – Diagrama do sistema desenvolvido, com a integração entre as ferramentas selecionadas



Fonte: O autor

The Zeek Network Security Monitor

A primeira etapa é responsável por efetuar o monitoramento a coleta do fluxo de dados em tempo real do tráfego da rede, isto é, conforme novas conexões são realizadas, as mesmas devem ter suas informações coletadas e disponibilizadas para processamento. Para tal fim é aplicado o Zeek, um *framework open-source* de segurança de rede, que tem como principal característica o monitoramento de todo o tráfego de rede, onde o fluxo monitorado é registrado em um extenso conjunto de *logs*, que incluem diversos protocolos.

Apache Kafka

Para realizar o transporte dos dados coletados pelo Zeek para o posterior processamento, utiliza-se o Apache Kafka, uma ferramenta distribuída que possui a capacidade de receber e distribuir um fluxo de dados como uma fila em tempo real. Além disso, possui baixa latência de resposta e altas taxas de transmissão (GÜRCAN; BERIGEL, 2018). As aplicações de Kafka conseguem tratar milhares de mensagens em intervalos reduzidos de tempo, e conseguem manter desempenho constante, mesmo com grandes volumes de dados (SOBREIRO, 2018).

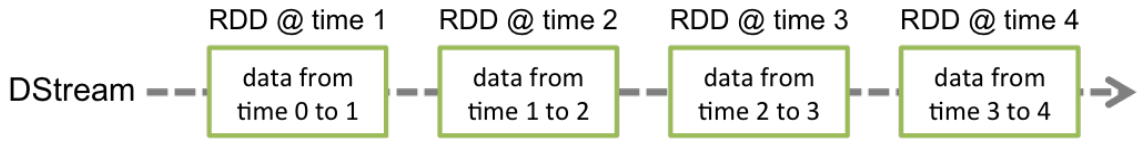
O Kafka utiliza o conceito de *producers* e *consumers*. Nesse cenário, consideramos o Zeek como *producer*, o qual efetua a escrita dos dados. Enquanto o Spark Streaming atua como *consumer*, lendo os dados conforme novas informações são escritas. Assim, abstrai-se a necessidade de *consumer* e *producer* terem conhecimento um do outro.

Apache Spark Streaming

A Plataforma Apache Spark é uma plataforma para processamento distribuído de dados, escrita em Java e Scala. A extensão Spark Streaming, baseada em micro-lotes, foi desenvolvida em 2014 como uma biblioteca executando no topo do Spark Engine para realizar analítica em tempo real (ZAHARIA et al., 2013).

Spark Streaming permite o processamento de fluxo de dados, para efetuar esse processo o mesmo recebe o fluxo e divide os dados em pequenos lotes que são processados pelo Spark. Esses pequenos lotes internamente são representados por uma sequência de Resilient Distributed Datasets (RDDs), conforme pode ser observado na Figura 3.3. O Spark Streaming possui uma abstração de alto nível denominada *Discretized Stream* (DStream), que representa um fluxo contínuo de dados e pode ser criada a partir de fluxo de dados de entrada, nesse caso o Kafka ou a partir de outro DStream (HAAS et al., 2019).

Figura 3.3 – Abstração do DStream



Fonte: Apache Spark: https://spark.apache.org/

Apache Cassandra

O Apache Cassandra é um banco de dados NoSQL que fornece um sistema de armazenamento de código aberto, distribuído e descentralizado para gerenciar grandes volumes de dados estruturados espalhados pelo mundo. É um banco de dados escalável, tolerante a falhas e consistente, orientado a colunas. O Cassandra fornece suporte robusto para *clusters* de dados que abrangem vários *datacenters* com replicação assíncrona, tolerando processos de baixa latência para todos os clientes. Especialmente nos últimos tempos, o mesmo está sendo utilizado por grandes corporações, como Facebook, Twitter, Cisco, Ebay, em aplicativos de *streaming* de dados em tempo real (GÜRCAN; BERIGEL, 2018).

3.2 PROCESSAMENTO

A primeira tarefa realizada pelo sistema é a coleta do fluxo de dados da rede. O ZEEK é configurado para realizar o monitoramento de determinada rede, gerando *logs* com as informações coletadas. O arquivo de *log* gerado denominado “conn” pode ser considerado o arquivo principal, pois possui todas as conexões que foram realizadas utilizando os protocolos Internet *Control Message Protocol* (ICMP), *Transmission Control Protocol* (TCP), e *User Datagram Protocol* (UDP). Os principais campos da conexão presentes no arquivo “conn” são apresentados na Tabela 1.

Tabela 3.1 – Campos do log “conn”

Campos	**Descrição**
UID	Identificador exclusivo da conexão
ORIG_H	Endereço IP que originou a conexão
ORIG_P	Número da Porta que originou a conexão
RESP_H	Endereço IP que respondeu a conexão
RESP_P	Número da Porta que respondeu a conexão
PROTO	Identificação do protocolo da camada de transporte
SERVICE	Identificação do protocolo de aplicação
DURATION	Duração da conexão
TS	Timestamp referente ao primeiro pacote
ORIG_IP_BYTES	Número de Bytes que o IP de origem enviou
RESP_IP_BYTES	Número de Bytes que o IP de resposta enviou

Fonte: O autor

Posteriormente, os dados são enfileirados no Kafka, que foi configurado com 1 *Broker* e este com 1 Tópico, até que sejam consumidos pelo *Spark Streaming*, que realiza o consumo de acordo com o intervalo de tempo configurado, formando lotes de dados a serem processados.

O Spark cria um *DStream* que consome todas as mensagens neste intervalo de tempo, e este *DStream* contém uma sequência de RDDs. Para acessar os RDDs contidos no *DStream* utilizou-se a função “*foreachRDD*”. Com acesso ao RDD, que possui a coleção dos dados “.json”, efetuou-se sua transformação para um *dataframe*. Utilizando este *dataframe* é feita então uma filtragem, criando um novo *dataframe* apenas com os dados presentes no arquivo de *log* “conn”.

É utilizado então o módulo “pyspark.sql”, que possui funções como *select*, *groupBy*, *count*, e outras, para criar novos *dataframes* que já estejam no formato das tabelas que almeja-se salvar no Cassandra. Informando a tabela e o *keyspace* alvos, é feita então a inserção dos dados do *dataframe* no Cassandra. As tabelas que foram

criadas e seus respectivos campos utilizados estão descritos a seguir, na Tabela 3.2.

Tabela 3.2 – Tabelas e dados salvos no Cassandra

Tabela	**Campos utilizados**
proto_total	proto, count
proto_day	proto, timestamp(day), count
proto_hour	proto, timestamp(hour), count
service_day	service, timestamp(day), count
service_hour	service, timestamp(hour), count
flow_day	timestamp(day), orig_h, resp_h, resp_p, proto, count
flow_hour	timestamp(hour), orig_h, resp_h, resp_p, proto, count
orig_h_day	orig_h, timestamp(day), count
orig_h_hour	orig_h, timestamp(hour), count
resp_h_day	resp_h, timestamp(day), count
resp_h_hour	resp_h, timestamp(hour), count
resp_p_day	resp_p, timestamp(day), count
resp_p_hour	resp_p, timestamp(hour), count

Fonte: O autor

Para realizar o acúmulo do número de conexões em cada tabela, no campo “count” foi utilizado um tipo de dado próprio do Cassandra, denominado *Counter*, que realiza acúmulos automaticamente quando são inseridos novos dados na tabela. Uma vez que os dados estão salvos em suas respectivas tabelas, é possível fazer consultas sobre as mesmas e obter os resultados do processamento.

4 RESULTADOS EXPERIMENTAIS

Este capítulo está dividido em duas seções. Na primeira (4.1) é feita a descrição do conjunto de dados que foi processado. Na Seção 4.2, são expostos os resultados obtidos através da execução do sistema desenvolvido.

4.1 METODOLOGIA

Para a implementação e validação do sistema foi utilizada uma máquina física com os seguintes recursos de *hardware*:

- Processador AMD Ryzen 5 2600 3.4 gigahertz com 6 núcleos;
- 8 gigabytes de memória RAM (*Random Access Memory*);
- 1 terabyte de disco rígido.

O sistema operacional usado na máquina descrita foi o Ubuntu 18.04.3, e foram criados Docker *containers*[1] para cada uma das ferramentas de código aberto utilizadas. Assim, cada ferramenta é configurada separadamente em seu respectivo *container* e então é feita a integração entre elas.

Para validar o sistema, foi feito o monitoramento da rede local e utilizou-se um arquivo ".pcap" com 11 gigabytes de dados controlados para inserir na fila do Kafka mais de 320.000 conexões, simulando assim, uma grande quantidade de conexões repentinas na rede monitorada. Os resultados obtidos a partir da execução do sistema foram validados com base nas informações disponíveis sobre o conjunto de dados controlados utilizado.

O conjunto de dados utilizado foi o CICIDS2017[2], disponibilizado pela *University of New Brunswick* (UNB), por possuir um tráfego de rede normal e ataques mais atualizados que se assemelham aos dados verdadeiros. O conjunto possui a coleta de dados referente a cinco dias, onde cada dia está salvo em um arquivo ".pcap", sendo possível escolher os dados conforme o dia desejado. Para o experimento foi selecionado o dia 04/07, que tem um alto número de conexões das 8 às 17 horas e possui anomalias causadas por ataques de força bruta. Na Tabela 4.1 podem ser observados os períodos em que foram efetuados os ataques, bem como a descrição dos mesmos.

[1] Docker *containers*: https://www.docker.com/resources/what-container
[2] Conjunto de dados CICIDS2017: https://www.unb.ca/cic/datasets/ids-2017.html

Tabela 4.1 – Detalhes sobre as anomalias presentes no tráfego de rede selecionado

Período	Atacante	Vítima	Descrição
9:20 - 10:20	172.16.0.1	192.168.10.50	FTP-Patator
14:00 - 15:00	172.16.0.1	192.168.10.50	SSH-Patator

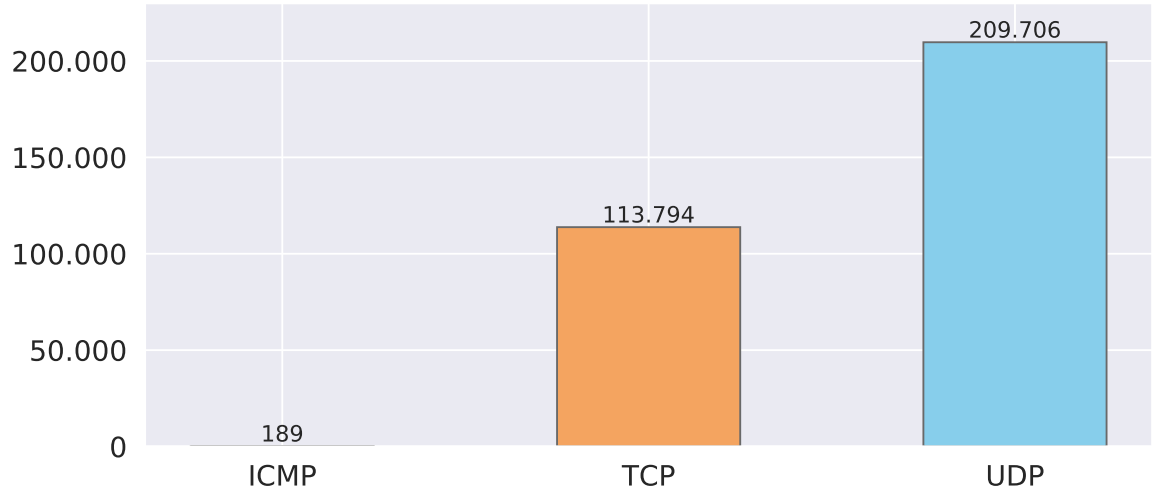
Fonte: O autor

4.2 RESULTADOS

Para a obtenção dos resultados exibidos nesta seção, utilizou-se os dados processados e salvos no Cassandra e foram gerados gráficos (manualmente) para facilitar a visualização dos dados obtidos. Tendo em vista que os dados utilizados são de um mesmo dia, o foco se deu em dividir o número de conexões por hora.

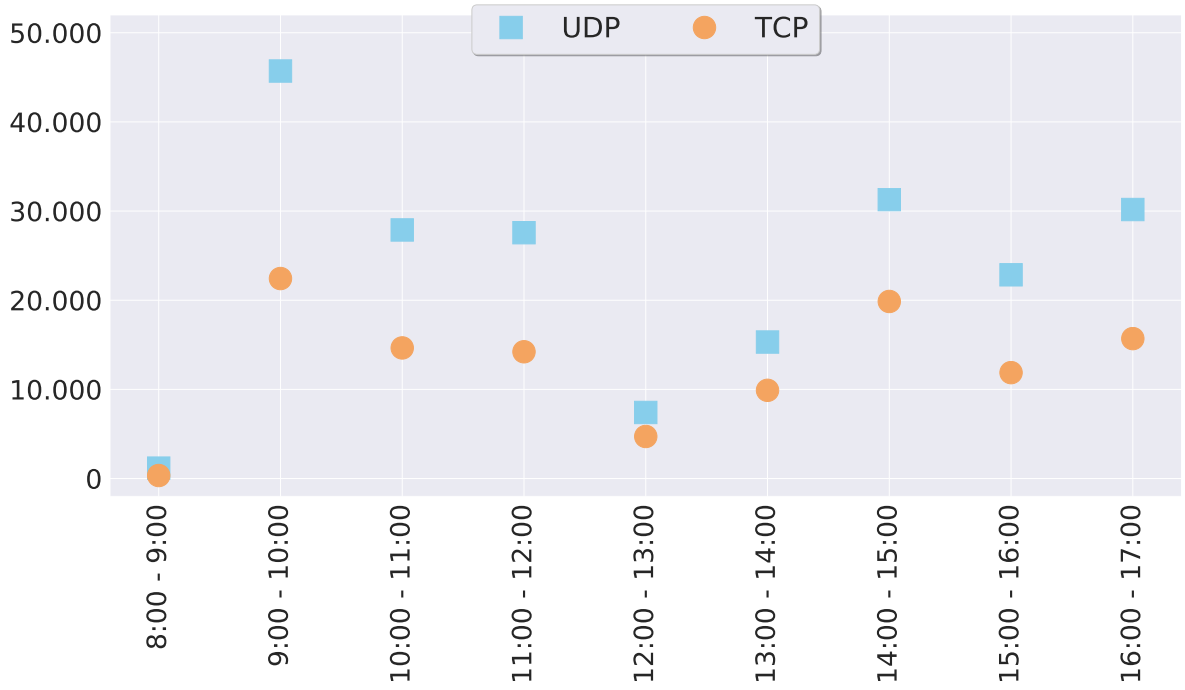
Uma vez inserido o conteúdo do arquivo ".pcap" na fila do Kafka para simular um alto número de conexões na rede monitorada, o sistema levou em média 1 minuto e 36 segundos (considerando 10 execuções) para processar um total aproximado de 323.689 conexões e salvar os resultados no Cassandra. As conexões estão divididas em três protocolos, ICMP, TCP, e UDP, e a quantidade de conexões utilizando cada protocolo pode ser observado na Figura 4.1. Já o total de conexões por hora e protocolo são exibidos na figura 4.2.

Figura 4.1 – Número de conexões por protocolo de transporte



Fonte: O autor

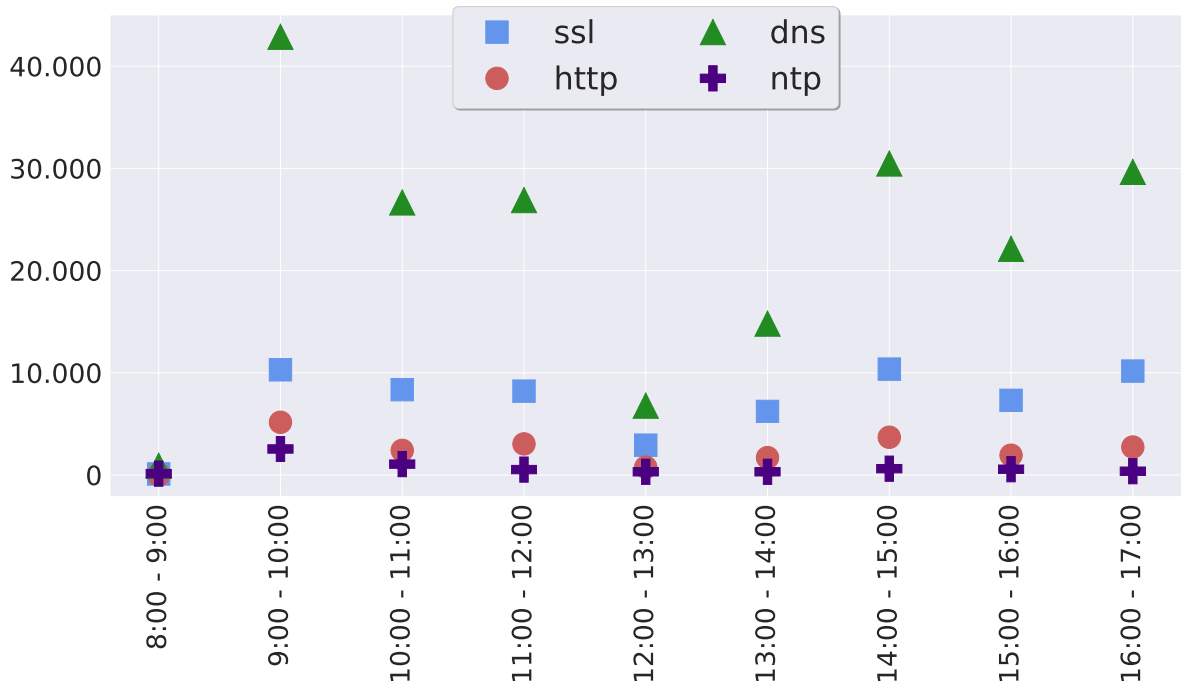
Figura 4.2 – Número de conexões por hora e protocolo de transporte



Fonte: O autor

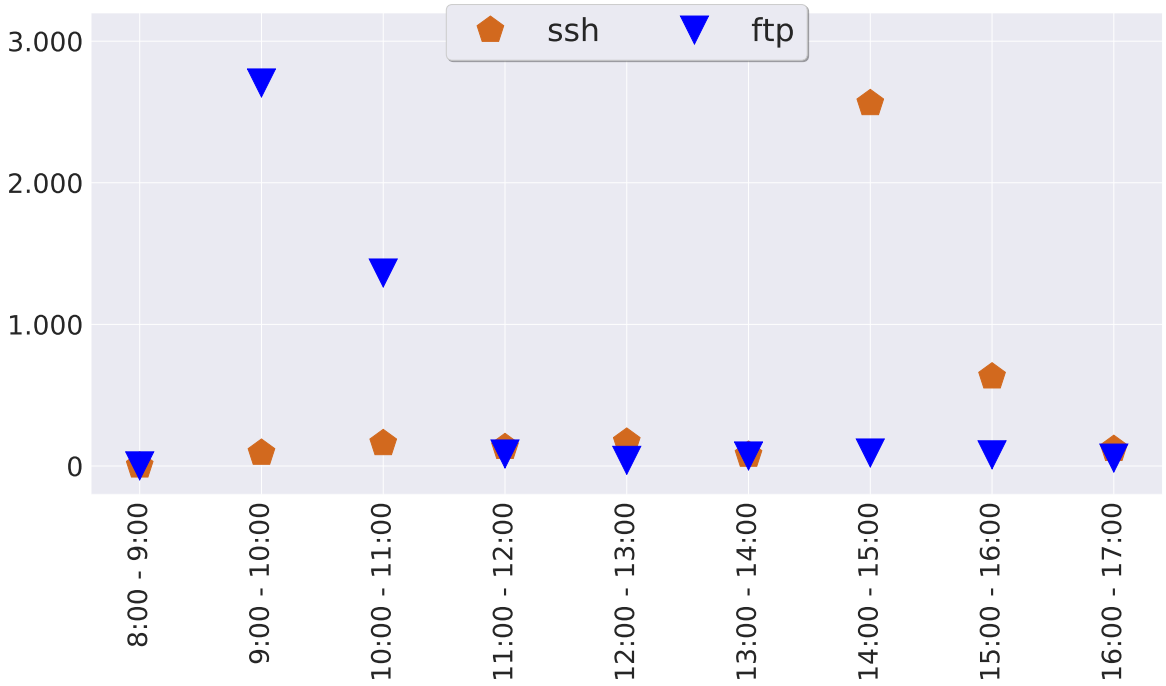
Deste total de conexões foram identificados diversos tipos de protocolos da camada de aplicação, como *Domain Name System* (DNS), *Hypertext Transfer Protocol* (HTTP), *Network Time Protocol* (NTP), *Secure Socket Layer* (SSL), *Secure Shell* (SSH), *File Transfer Protocol* (FTP) e outros. Na Figura 4.3, é possível observar o número de conexões por hora de alguns dos principais protocolos de aplicação. O maior número de conexões foi DNS, seguido por SSL e outros. Já a figura 4.4 exibe separadamente apenas os protocolos de aplicação utilizados pelo atacante, mostrando claramente um pico de conexões no horário dos ataques de força bruta.

Figura 4.3 – Número de conexões por hora e protocolo de aplicação



Fonte: O autor

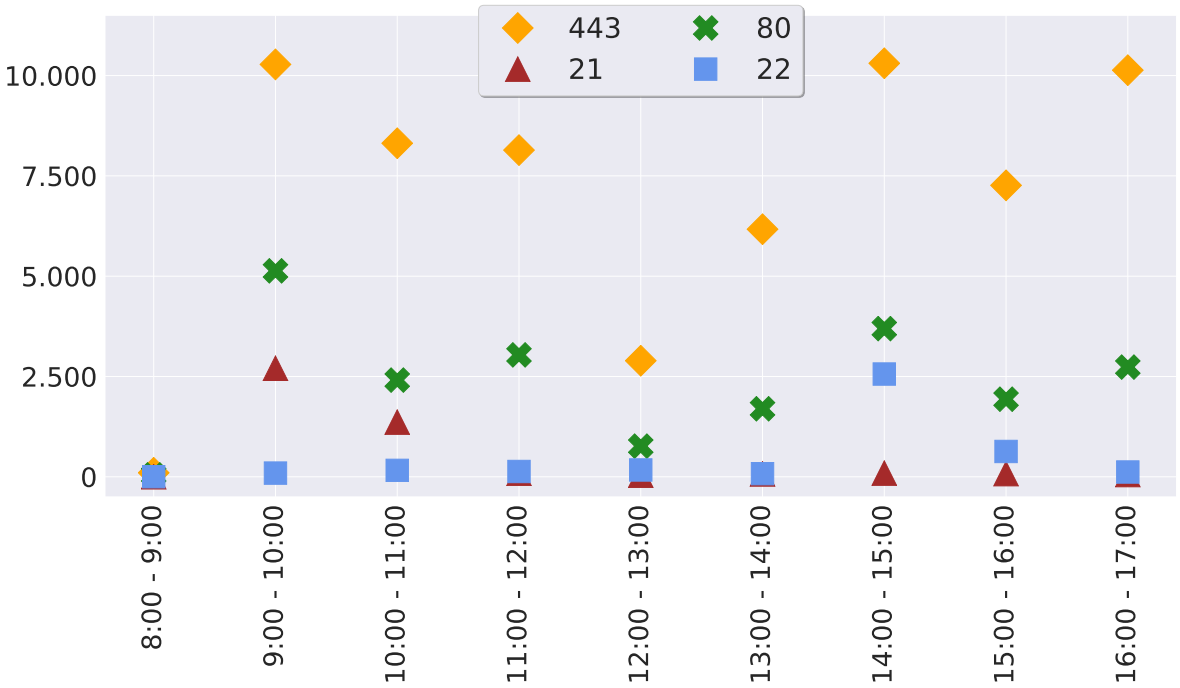
Figura 4.4 – Número de conexões por hora e protocolo de aplicação utilizado pelo atacante



Fonte: O autor

A ação do atacante pode ser igualmente observada analisando o número de conexões de cada uma das portas utilizadas pelos respondedores. A Figura 4.5 mostra o número de conexões por hora de algumas das portas mais utilizadas, às quais correspondem à diferentes protocolos de aplicação. A porta 21 corresponde ao FTP, a 22 ao SSH, a 443 ao *Hyper Text Transfer Protocol Secure* (HTTPS), e a 80 ao HTTP. É possível observar um aumento súbito no número de conexões da porta 21 das nove às onze horas, assim como na porta 22 das quatorze às quinze horas, que correspondem aos horários e protocolos de aplicação utilizados pelo atacante para realizar os ataques de força bruta.

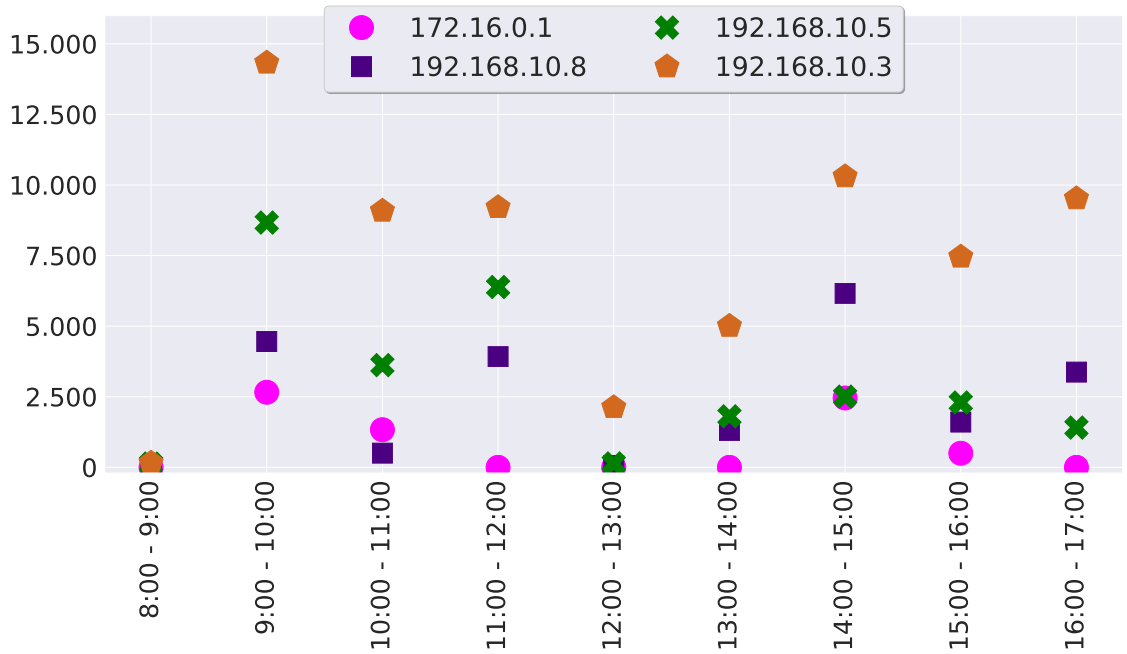
Figura 4.5 – Número de conexões por hora e porta do respondedor



Fonte: O autor

Outra informação disponibilizada pelo sistema é a quantidade de conexões por hora que foram originadas por um determinado IP. Como pode ser observado na Figura 4.6, que contém os IPs que mais originaram conexões, o IP do atacante “172.16.0.1”, que é tratado pelo *firewall*, passa a originar mais de duas mil conexões repentinamente no período dos ataques.

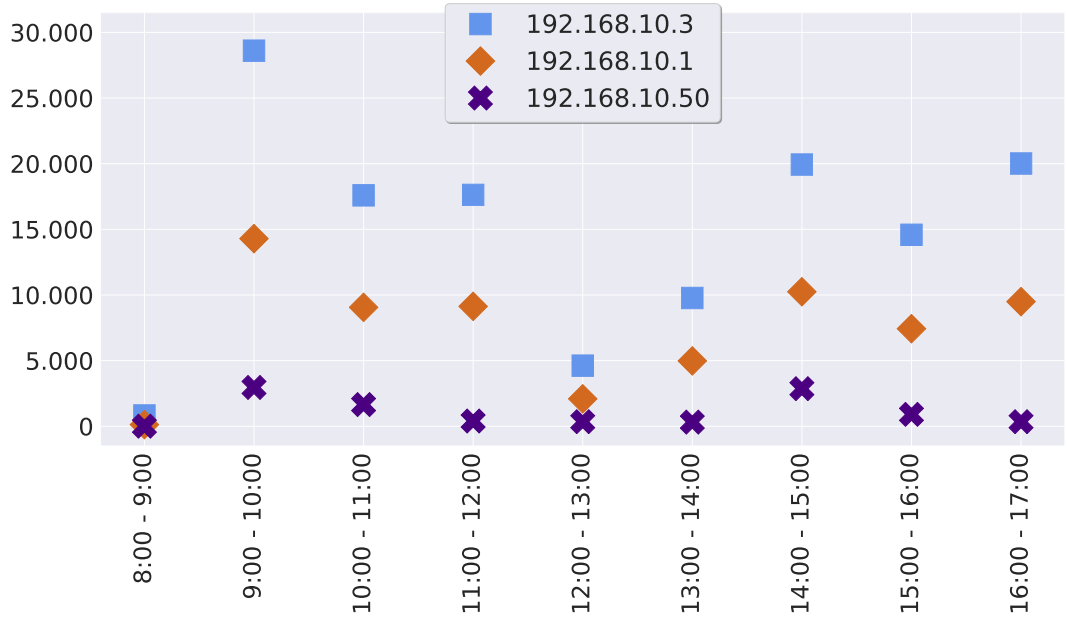
Figura 4.6 – Número de conexões por hora e IP originador



Fonte: O autor

A Figura 4.7, que apresenta informações sobre o número de conexões, divididas por hora, que foram respondidas por um determinado IP, evidencia que o IP “192.168.10.50” sofre um aumento súbito no número de conexões respondidas, que passam a ser de dezenas nos momentos sem ataque, para milhares nos períodos dos ataques.

Figura 4.7 – Número de conexões por hora e IP respondedor



Fonte: O autor

Para tornar possível análise de fluxo, o sistema acumula o número de conexões utilizando os seguintes campos:

- IP que originou a conexão;
- o IP que respondeu a conexão;
- A porta utilizada para responder a conexão;
- O protocolo de transporte utilizado;
- *Timestamp*.

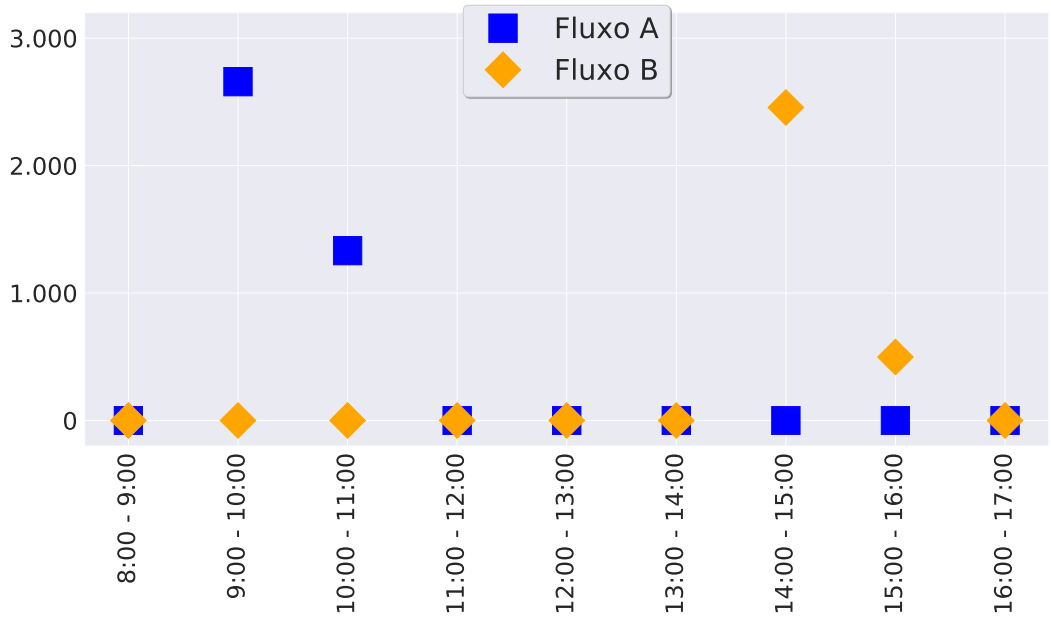
Ao analisar o fluxo gerado, ficaram evidentes as anomalias geradas pelo ataque. Ademais, o fluxo pode disponibilizar na mesma consulta várias informações relevantes, como neste caso o horário do ataque, o IP do atacante, o IP vítima do ataque, além da porta e o protocolo que foram utilizados. Na Figura 4.8 constam os fluxos com maior número de conexões (referenciados como “Fluxo A” e “Fluxo B”), que são os originados pelo atacante. Os Fluxos “A” e “B” estão detalhados na tabela a seguir (4.2).

Tabela 4.2 – Descrição dos fluxos “A” e “B”

Fluxo	IP do originador	IP do respondedor	Protocolo	Porta
A	172.16.0.1	192.168.10.50	TCP	21
B	172.16.0.1	192.168.10.50	TCP	22

Fonte: O autor

Figura 4.8 – Número de conexões por hora e fluxos do atacante



Fonte: O autor

Conforme pôde ser visto neste capítulo, o sistema desenvolvido obteve os resultados esperados, tendo processado, através da simulação feita, mais de 300.000 conexões súbitas. Com uso das informações salvas no Cassandra oriundas do processamento efetuado, foi possível realizar consultas e assim perceber diferentes anomalias que foram causadas pelos ataques de força bruta. É importante ressaltar que as ferramentas utilizadas no sistema desenvolvido são escaláveis, o que viabiliza a aplicação do sistema sobre um conjunto de dados maior.

5 CONCLUSÃO

Os avanços dos serviços providos pelas redes de computadores têm assumido um papel importante na vida contemporânea. No cotidiano, muitas vezes acabamos dependentes da rede, seja em atividades pessoais, profissionais ou acadêmicas. Nesse contexto, garantir a integridade do funcionamento da rede é uma tarefa que se torna cada vez mais essencial, e sobretudo complexa, dado o grande crescimento do número de dispositivos conectados e dados gerados nos últimos anos.

O presente trabalho desenvolveu um sistema, com base na camada *Speed* da arquitetura Lambda, para efetuar o processamento do fluxo de dados da rede em tempo real, disponibilizando informações que possibilitam o monitoramento da rede. Utilizou-se diferentes ferramentas de código aberto que foram arquitetadas para lidar com um grande volume de dados, resultando assim em um sistema escalável, o que é imprescindível visto que a quantidade de dados gerados na rede tende a crescer ainda mais nos próximos anos. O sistema foi projetado para que futuramente possa ser integrado com o trabalho proposto por (HAAS et al., 2019), que aborda um sistema baseado na arquitetura Lambda, mas não possui a camada Speed implementada na íntegra.

Através da simulação feita com o uso do *dataset* disponibilizado pela *University of New Brunswick*, foi possível validar o funcionamento da estrutura lógica e o desempenho do sistema, bem como a integração entre as ferramentas escolhidas. O sistema obteve os resultados esperados, tendo processado mais de 320.000 conexões e sendo possível notar, com o uso das informações oriundas do processamento e que foram salvas no Cassandra, as anomalias presentes no conjunto de dados.

Como trabalhos futuros sugere-se adicionar ao sistema algoritmos de aprendizado de máquina, para que após feito o processamento dos dados, seja possível detectar anomalias no fluxo de dados da rede de maneira automática. Por fim, existe a intenção de testar o sistema operando sobre um conjunto grande de dados.

REFERÊNCIAS BIBLIOGRÁFICAS

BÄR, A. et al. Large-scale network traffic monitoring with dbstream, a system for rolling big data analysis. In: IEEE. **2014 IEEE International Conference on Big Data (Big Data)**. [S.l.], 2014. p. 165–170.

CARTER, P. Big data analytics: Future architectures, skills and roadmaps for the cio. **IDC White Papers**, 2011.

CLAY, P. A modern threat response framework. **Network Security**, Elsevier, v. 2015, n. 4, p. 5–10, 2015.

DUMBILL, E. What is big data? 2012. Disponível em: <https://www.oreilly.com/radar/what-is-big-data/>. Acesso em: 08 out. 2019.

GÜRCAN, F.; BERIGEL, M. Real-time processing of big data streams: Lifecycle, tools, tasks, and challenges. In: IEEE. **2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)**. [S.l.], 2018. p. 1–6.

HAAS, A. et al. Um sistema por processamento de fluxos aplicado à análise e monitoramento da rede. Universidade Federal de Santa Maria, 2019.

HAN, S. H.; NASRIDINOV, A.; RYU, K. H. Information flow monitoring system. **IEEE Access**, IEEE, v. 6, p. 23820–23827, 2018.

KOZIK, R. Distributed system for botnet traffic analysis and anomaly detection. In: IEEE. **2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)**. [S.l.], 2017. p. 330–335.

KREPS, J. Questioning the lambda architecture. **Online article, July**, 2014. Disponível em: <https://www.oreilly.com/radar/questioning-the-lambda-architecture/>. Acesso em: 08 out. 2019.

LEE, Y.; LEE, Y. Toward scalable internet traffic measurement and analysis with hadoop. **ACM SIGCOMM Computer Communication Review**, ACM, v. 43, n. 1, p. 5–13, 2013.

LOBATO, A. G. P.; LOPEZ, M. A.; DUARTE, O. Um sistema acurado de detecção de ameaças em tempo real por processamento de fluxos. **XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'2016), Salvador, Bahia**, 2016.

LOPEZ, M. A. et al. Aprendizado de máquina em plataformas de processamento distribuído de fluxo: Análise e detecção de ameaças em tempo real. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC**, v. 2018, p. 150–206, 2018.

LOPEZ, M. E. A. **A monitoring and threat detection system using stream processing as a virtual function for Big Data**. 2018. Tese (Doutorado) — Sorbonne Université; Universidade federal do Rio de Janeiro, 2018.

MARZ, N. How to beat the cap theorem. **Thoughts from the Red Planet**, 2011. Disponível em: <http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html>. Acesso em: 08 out. 2019.

MARZ, N.; WARREN, J. **Big Data: Principles and best practices of scalable real-time data systems**. [S.l.]: New York; Manning Publications Co., 2015.

OUNACER, S. et al. A new architecture for real time data stream processing. **INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS**, SCIENCE & INFORMATION SAI ORGANIZATION LTD 19 BOLLING RD, BRADFORD, WEST . . . , v. 8, n. 11, p. 44–51, 2017.

SEYVET, N.; VIELA, I. M. Applying the kappa architecture in the telco industry. **Online article, May**, 2016. Disponível em: <https://www.oreilly.com/ideas/applying-the-kappa-architecture-in-the-telco-industry>. Acesso em: 08 out. 2019.

SILVA, E. d. C. d. Detecção de ataques de negação de serviço utilizando aprendizado de máquina. 2016.

SOBREIRO, S. A. R. **Estudo de tecnologias para sistemas de Big Data**. 2018. Tese (Doutorado), 2018.

SOUMAYA, O. et al. Real-time data stream processing challenges and perspectives. **International Journal of Computer Science Issues (IJCSI)**, International Journal of Computer Science Issues (IJCSI), v. 14, n. 5, p. 6–12, 2017.

STOREY, V. C.; SONG, I.-Y. Big data technologies and management: What conceptual modeling can do. **Data & Knowledge Engineering**, Elsevier, v. 108, p. 50–67, 2017.

ZAHARIA, M. et al. Discretized streams: Fault-tolerant streaming computation at scale. In: ACM. **Proceedings of the twenty-fourth ACM symposium on operating systems principles**. [S.l.], 2013. p. 423–438.

ZARPELÃO, B. B. et al. Detecção de anomalias em redes de computadores. [sn], 2010.