

**UNIVERSIDADE FEDERAL DE SÃO CARLOS**

**CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA**

**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**UMA IMPLEMENTAÇÃO PARALELA DO  
ALGORITMO DE DETECÇÃO DE NOVIDADE  
EM STREAMS MINAS**

**LUÍS HENRIQUE PUHL DE SOUZA**

**ORIENTADOR: PROF. DR. HERMES SENGER**

São Carlos – SP

Fevereiro/2020

**UNIVERSIDADE FEDERAL DE SÃO CARLOS**

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**UMA IMPLEMENTAÇÃO PARALELA DO  
ALGORITMO DE DETECÇÃO DE NOVIDADE  
EM STREAMS MINAS**

**LUÍS HENRIQUE PUHL DE SOUZA**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas de Computação

Orientador: Prof. Dr. Hermes Senger

São Carlos – SP

Fevereiro/2020

# SUMÁRIO

|  |           |
|--|-----------|
| <b>CAPÍTULO 1 – INTRODUÇÃO</b>                             | <b>4</b>  |
| 1.1 Objetivos . . . . .                                    | 5         |
| 1.2 Motivação . . . . .                                    | 5         |
| 1.3 Proposta Metodológica . . . . .                        | 6         |
| <b>CAPÍTULO 2 – FUNDAMENTOS CIENTÍFICOS E TECNOLÓGICOS</b> | <b>7</b>  |
| 2.1 Computação em Nuvem, Borda e Névoa . . . . .           | 7         |
| 2.1.1 Computação em Nuvem . . . . .                        | 7         |
| 2.1.2 Computação de Borda . . . . .                        | 8         |
| 2.1.3 Computação em Névoa . . . . .                        | 10        |
| 2.2 Mineração de Dados e Fluxo de Dados . . . . .          | 10        |
| 2.3 Plataformas de processamento distribuído . . . . .     | 10        |
| 2.4 Apache Flink . . . . .                                 | 12        |
| 2.5 Detecção de Novidade . . . . .                         | 12        |
| 2.6 O algoritmo MINAS . . . . .                            | 12        |
| <b>CAPÍTULO 3 – TRABALHOS RELACIONADOS</b>                 | <b>14</b> |
| 3.1 BigFlow . . . . .                                      | 14        |
| <b>CAPÍTULO 4 – IMPLEMENTAÇÃO E TESTES</b>                 | <b>16</b> |
| 4.1 Descrição da Implementação . . . . .                   | 16        |

|                                |                                     |           |
|--------------------------------|-------------------------------------|-----------|
| 4.2                            | Cenário de Teste . . . . .          | 16        |
| 4.3                            | Experimentos e Resultados . . . . . | 16        |
| <b>CAPÍTULO 5 – CRONOGRAMA</b> |                                     | <b>18</b> |
| 5.1                            | Cronograma . . . . .                | 18        |
| <b>REFERÊNCIAS</b>             |                                     | <b>19</b> |

# Capítulo 1

## INTRODUÇÃO

---

Paralelize o minas no flink. (não se preocupe com o uso, seja ele NIDS ou qualquer outra coisa) Use a detecção de intrusão apenas como validação do algoritmo.

cap 1: - Objetivo paralelizar minas em plataforma de big-data capaz de consumir streams de forma eficiente - Motivação: Minas é recente, com potencial em várias aplicações, por exemplo (NIDS, sensores, ...) para isso deseja-se uma implementação eficiente (low power, ou usar todo hardware) e escalável (big-data)

A Internet das Coisas (*Internet of Things* - IoT) é um tema frequentemente abordado em pesquisas de segurança em redes de computadores claramente motivado pelo crescimento dessa rede, estimada em 8.4 bilhões de dispositivos em 2017 [1] e, pelo risco que ela representa fundamentado no histórico de ataques massivos realizados por milhares de nós subvertidos como o realizado pela *botnet* mirai em 2016 [2]. Mais preocupante nesse cenário são os fatores que possibilitaram esses ataques: falta de controle sobre a origem do hardware e software embarcado nos dispositivos além das cruciais atualizações de segurança.

Com esse desafio de segurança, especialmente em IoT industrial onde a subversão de dispositivos pode causar danos reais imediatamente, profissionais de segurança de redes e operadores de redes são confrontados com enorme superfície de ataque composta por diversos sistemas, tecnologias com longo tempo de vida e poucas atualizações, um sistema de detecção de intrusão (*Intrusion DetectionSystem* - IDS) operando na rede de computadores local torna-se uma ferramenta muito importante para defesa dessa rede e os serviços que ela suporta.

Os IDS foram tradicionalmente construídos à partir de técnicas de mineração de dados (*Data Mining* - DM), aprendizado de máquina (*Machine Learning* - ML), mais especificamente Detecção de Novidades (*Novelty Detection* - ND) para detectar ataques e descobrir novos padrões, porém ao analisar tráfego de rede a velocidade da análise deve ser próxima ou superior

à velocidade da rede analisada além de não consumir mais recursos do que a própria rede analisada. Mais restrições nesse sentido devem ser incorporadas quando trata-se de uma rede IoT, diferente de uma rede operando em computação na nuvem (*Cloud Computing* - cloud), especialmente latência e banda são ainda mais restritos e ao mitigar esses atributos movendo o IDS para o mais próximo da rede IoT passando a processar na névoa computacional (*Fog Computing* - fog) armazenamento e processamento são também restringidos. Portanto uma única leitura do conjunto analisado, rápida atualização e distribuição do modelo de detecção e resultados em tempo real são características positivas encontradas nas técnicas de mineração e processamento de fluxos de dados (*Data Streams* - DS).

Nesse contexto, foca-se na arquitetura de IDS proposta por [3] baseada no algoritmo de ND em DS MINAS [4] e na implementação BigFlow [5] proposta para redes 10 *Gigabit Ethernet*. Acredita-se que a fusão dessas abordagens em uma nova implementação seja capaz de tratar uma rede de maior fluxo com nível comparável de precisão da análise com o mesmo hardware e maior capacidade de escalonamento horizontal com distribuição de carga entre nós na Fog.

Ideias:

Necessidade de utilizar recursos da Fog para eliminar latência de comunicação com a nuvem (enviar os dados para classificar na nuvem teria alta latência)

Necessidade de processamento concorrente e distribuído para ter maior escalabilidade

## 1.1 Objetivos

Avaliar o algoritmo minas na plataforma Flink para detecção de intrusão em redes IoT.

Objetivos secundários: Identificar métricas na literatura: - Quanto a detecção de anomalias; - e desempenho na detecção de intrusão da literatura; Implementar Minas sobre Apache Flink para detecção de intrusão em redes IoT; Extrair as métricas de detecção de anomalias da implementação com datasets diferentes; Validar a implementação comparando as métricas extraídas da implementação com as encontradas na literatura; Extrair as métricas de desempenho da implementação com datasets diferentes;

## 1.2 Motivação

- flink não foi abordado para intrusões em redes iot; - contribuição de técnica/método para a área de segurança da informação;

Implementar a arquitetura proposta em [3] (IDS-IoT) com foco no volume de fluxo explorando aspectos de paralelismo e distribuição de recursos em fog e cloud visando alta vazão e baixa latência.

Para redução da latência pretende-se utilizar recursos próximos à rede monitorada (fog) distribuindo o volume entre vários nós (com recursos muitas vezes limitados) para dar vazão ao processamento. Além disso, para tarefas com maiores exigências de recursos de processamento e armazenamento como treinamento e reconstrução do modelo pretende-se utilizar computação em nuvem.

Implementar o algoritmo MINAS [4] buscando máximo desempenho nos aspectos latência e vazão utilizando recursos de fog e nuvem além de avaliar seus resultados qualitativos na detecção e classificação de novidades em *datasets* relevantes a detecção de intrusão em redes IoT.

## 1.3 Proposta Metodológica

Implementar – e avaliar esta implementação – o algoritmo MINAS [4] aplicando técnicas, como Apache Flink, usadas na solução BigFlow [5]; Aplicar essa implementação na arquitetura proposta por [3] e comparar o desempenho com a implementação e conjunto de dados (dataset) original; Avaliar o desempenho e escalabilidade da aplicação com dataset MAWIFlow.

Flink - dizer que vai usar, para fazer o que e porque escolheu

Kafka ?

Raspberry - para todos

A base -

# Capítulo 2

## FUNDAMENTOS CIENTÍFICOS E TECNOLÓGICOS

---

### 2.1 Computação em Nuvem, Borda e Névoa

O modelo de Computação em Nuvem (*Cloud Computing*) permite alocar recursos como redes, servidores, armazenamento, aplicações e serviços de maneira conveniente e seu provisionamento ágil concede elasticidade para atender demandas variáveis com custo mínimo (MELL; GRANCE, 2012).

#### 2.1.1 Computação em Nuvem

Computação em Nuvem (em inglês, *Cloud Computing*) habilita o acesso através da rede e sob demanda a um grupo compartilhado de recursos de computação configuráveis (servidores, redes, aplicativo, armazenamento, serviços, etc.) que podem ser provisionados ou liberados rapidamente com o mínimo esforço de gerenciamento ou interação com o provedor de serviços (MELL; GRANCE, 2012). As principais características do *cloud computing* são:

Alternativamente, a Computação na Borda (*Edge Computing*) destaca-se no processamento em tempo real de dados originários da própria borda além de atender preocupações de segurança e privacidade (SHI et al., 2016).

- **Serviço sob Demanda:** o cliente pode provisionar capacidades de computação (ex: tempo de servidor e armazenamento) conforme o necessário sem requer interação com o provedor de serviço;
- **Amplo acesso à rede:** o acesso aos recursos de computação e capacidades ocorre pela rede através de mecanismos padrões que permitem o acesso por plataformas heterogêneas (celulares, computadores, tablets, etc.)



- **Agrupamento de recursos:** para servir múltiplos clientes, os recursos de computação são agrupados usando o modelo *multi-tenancy* com recursos físicos e virtuais diferentes dinamicamente atribuídos e reatribuídos de acordo com a demanda do cliente;
- **Elasticidade:** as capacidades de computação são rapidamente provisionadas ou liberadas, em alguns casos automaticamente, para escalar conforme a demanda;
- **Serviço mensurado:** os recursos de computação são monitorados, controlados e reportados fornecendo transparência para o provedor de serviços e para o cliente sobre as capacidades que foram consumidas.

Segundo, Mell e Grance (2012), a implantação da Computação em Nuvem pode ocorrer através dos seguintes modelos:

- **Nuvem privada:** a infraestrutura da nuvem é provisionada e dedicada para um único cliente ou organização. Nesse modelo, o cliente gerencia e controla a infraestrutura, ou pode delegar essas tarefas a uma empresa terceira. A infraestrutura pode estar dentro ou fora das instalações da organização;
- **Nuvem comunitária:** a infraestrutura de nuvem é fornecida para um grupo exclusivo de clientes que compartilham de um mesmo interesse (requerimentos de segurança, desempenho, políticas, etc.). Esse tipo de nuvem pode ser gerenciado pelo próprio grupo, ou organizações terceiras, podendo estar dentro ou fora das instalações da empresa;
- **Nuvem pública:** a infraestrutura da nuvem é provisionada e oferecida para uso público. É gerenciado e operado por um provedor de nuvem.
- **Nuvem híbrida:** a infraestrutura desse tipo de nuvem é uma composição de duas ou mais modelos de implantação de *cloud* (privada, pública e comunitária) que formam um entidade única e são unidos por tecnologias padronizadas que habilitam a portabilidade de dados e aplicações.

### 2.1.2 Computação de Borda

A computação de borda (em inglês, *Edge Computing*) refere-se às tecnologias que permitem que a computação seja executada na borda da rede. Define-se borda ou *edge* como qualquer recurso de computação e de rede ao longo do caminho entre as fontes de dados e os data centers da nuvem (SHI et al., 2016). Por exemplo, um *gateway* numa casa é o borda entre os dispositivos

IoT e a nuvem. Na borda, é possível fazer armazenamento, processamento e descarregamento de dados, assim como distribuir as requisições e entregar os serviços das nuvens aos usuários. Shi et al. (2016) ressalta que essas capacidades dentre outras dos nós da borda (*edge nodes*) possibilitam que a computação de borda: reduzir latência na resposta da nuvem, pois o dado pode ser processado na borda; enviar somente os dados necessários, aproveitando melhor a banda e a transmissão de dados, e também consumindo menos recursos de computação na nuvem; aumentar a privacidade dos dados, pois o dado pode ser processado na borda.

A computação de borda tenta trazer a computação para mais próximo das fontes de dados, como mostra a Figura 2.1. Como é observado na figura, os componentes desse tipo de computação podem ser tanto produtores como consumidores, não só requisitando serviços e conteúdo da nuvem, mas também realizando tarefas da nuvem. Algumas aplicações da computação de borda são: nós de borda fazendo análise de vídeo; descarregar a nuvem de parte da computação, pois a borda também tem capacidade de processamento; privacidade dos dados produzidos por *smart homes* e sensores IoT, pois dados sensíveis não precisam ir para a nuvem, liberando também cardas de dados na rede; processamento distribuído nos nós da borda de dados provenientes de milhões de sensores em cidades inteligentes.

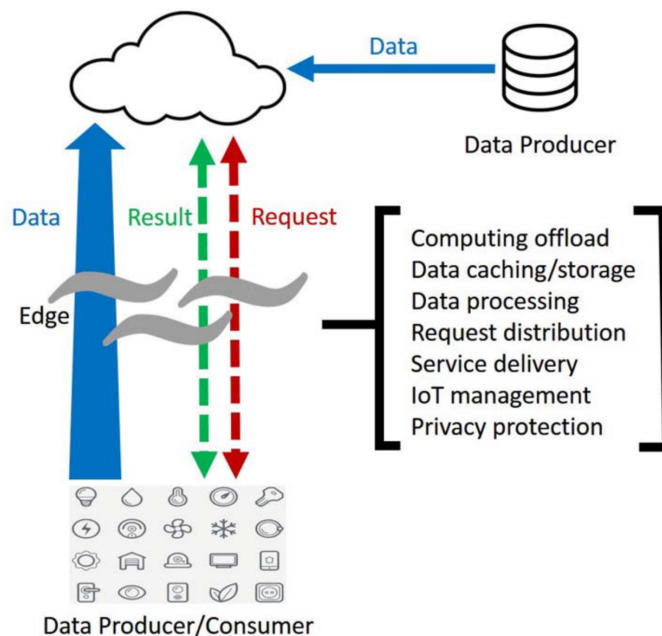


Figura 2.1: Paradigma de *Edge Computing* (SHI et al., 2016).

### 2.1.3 Computação em Névoa

## 2.2 Mineração de Dados e Fluxo de Dados

Mineração de Dados é o processo de descoberta de padrões em conjuntos de dados utilizando métodos derivados de aprendizagem de máquina, estatística e banco de dados (GABER; ZASLAVSKY; KRISHNASWAMY, 2005). Um caso de mineração de dados é *Big Data* onde o conjunto de dados não pode ser processado em um tempo viável devido a limitações como armazenado na memória ou armazenamento principal.

Além da dimensão de armazenamento outra dimensão que afeta a maneira como dados são modelados e manipulados é o tempo. Um Fluxo de Dados (*Data Stream*) é uma sequência de registros produzidos a uma taxa muito alta, associadas ao tempo real, ilimitados, que excede recursos de armazenamento e comunicação (GABER; ZASLAVSKY; KRISHNASWAMY, 2005). Modelos de mineração de fluxo de dados atendem a esses desafios utilizando restrições como apenas uma leitura do conjunto de dados e complexidade de processamento menor que linear (GAMA; RODRIGUES, 2007; GABER; ZASLAVSKY; KRISHNASWAMY, 2005).

## 2.3 Plataformas de processamento distribuído

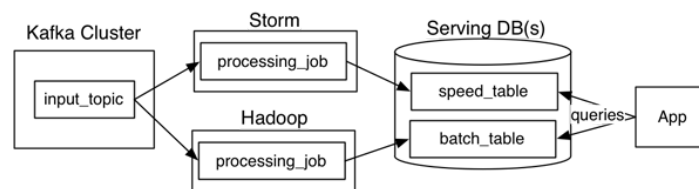
Tradicionalmente, aplicações foram construídas com um sistema gerenciador de banco de dados (SGBD) relacional ou não-relacional associado. Essa arquitetura, nomeada de arquitetura totalmente incremental por Marz e Warren (2015), foi evoluída e simplificada iterativamente durante décadas de uso porém ela não é adequada para sistemas em tempo real, como os sistemas de fluxo de dados.

O volume e velocidade de dados em um *Data Stream* leva a necessidade de distribuir o processamento acrescentando poder computacional a cada nó adicionado porém desafios como comunicação eficiente e sincronização de estado entre os nós assim como tolerância a falhas aumentam a complexidade de construção de um sistema distribuído em relação a um sistema tradicional. Para mitigar esses problemas foram propostas arquiteturas de processamento de fluxo de dados distribuído como arquitetura *lambda* e *kappa* além de diversas plataformas tanto de *Big Data* com características de tempo real como especializadas em fluxo de dados.

Arquitetura *lambda* divide o processamento em três camadas: lotes, serviço e velocidade. A camada de lotes atua sobre o conjunto mestre em modo de leitura sequencial, armazenando-o em sistema de arquivos distribuído e pré-processando várias visões sobre esse conjunto mestre,

essas visões (armazenadas num SGBD tradicional) são consumidas pela camada de serviço, que portanto tem acesso regular (leitura aleatória), no entanto as garantias oferecidas pela camada de lotes (escalabilidade, Corretude, tolerância a falhas) não atendem os requisitos de latência em um sistema em tempo real, portanto a camada de velocidade complementa os dados das visões com dados diretamente do conjunto mestre em tempo real diretamente para a camada de serviço (MARZ; WARREN, 2015).

A Figura 2.2 ilustra uma implementação da arquitetura *Lambda* onde a *Apache Kafka*, *Apache Storm*, *Apache Hadoop* implementam o conjunto mestre, camada de velocidade e camada de lotes respectivamente.



**Figura 2.2: Arquitetura *Lambda* com detalhes práticos (KREPS, 2014).**

No entanto, observações práticas de Kreps (2014) mostram que como o sistema de fila de mensagens (no exemplo *Apache Kafka*) fonte do conjunto de dados mestre por lidar com diversas fontes, garantir tolerância a falhas, replicação e armazenamento de longo prazo, poderia ser usado como fonte de verdade em conjunto com a arquitetura totalmente incremental (tradicional) substituindo as camadas de lotes e velocidade e o sistema de arquivos distribuído simplificando a aplicação de três implementações para duas eliminando a repetição de tarefas executadas pelas camadas de lotes e velocidade que produziam o mesmo resultado.

Essa arquitetura que limita-se ao processamento de *Stream* apenas *on-line* destila outras observações quanto a complexidade temporal do sistema completo de que se uma aplicação não processa os dados em tempo menor que linear, dados serão perdidos ou a complexidade espacial não será satisfeita, em resumo, a aplicação sempre deve ser mais rápida que o *Stream* (MARZ; WARREN, 2015).

- MapReduce, Hadoop, Spark, Storm

O que são e para que servem essas ferramentas

Breve descrição do MapReduce e Hadoop

Breve descrição do Spark

Apache Spark é um framework de código fonte aberto para computação distribuída.[1] Foi desenvolvido no AMPLab da Universidade da Califórnia[2] e posteriormente repassado para a

Apache Software Foundation[3] que o mantém desde então. Spark provê uma interface para programação de clusters com paralelismo e tolerância a falhas.

---

Apache Spark citeZaharia é um *framework* para construção de sistemas de computação distribuída em *cluster* com garantias de tolerância a falhas (execução em computadores não confiáveis) utilizando como premissas: paralelização e localidade de dados, como

api em Python (dataframe de pandas)

## 2.4 Apache Flink

Breve descrição do Flink (como esse vai ser usado, precisa explicar um pouco melhor - 2 paginas pelo menos):

- arquitetura
- modelo de programacao
- 1 pequeno exemplo de codigo explicando

(LOPEZ, 2018)

## 2.5 Detecção de Novidade

Novelty Detection

breve descrição do que são algoritmos para DN

ver se tem algum survey e citar

## 2.6 O algoritmo MINAS

Breve descrição do MINAS (??)

ver paper da profa. Elaine

Detecção de intrusão em redes - riscos de segurança - técnicas de intrusão e tipos de ataques - mecanismo de detecção (análise de fluxo de rede -> detecção de anomalia) Detecção de novidades - técnicas de Detecção de novidades - MINAS (incluir métricas) - BigFlow (incluir métricas) Processamento de Streams (big data) - cloud? - redes como stream - Atraso -

---

Kafka/Spark/Flink Redes IoT - Restrição hardware (Energia, CPU, Mem, Rede) - Consideração FOG vs Cloud

# Capítulo 3

## TRABALHOS RELACIONADOS

---

---

cap 3: Trabalhos relacionados - Artigos sobre o Minas - outros que paralelizaram algoritmos de mineração de dados/streams alguns online (5-10 refs) - implementação paralelas/distribuídas em dispositivos pequenos

(LOPEZ, 2018)

Aqueles que contenham: - detecção de anomalia em streams - detecção de intrusão em rede com processamento de streams - BigFlow

### 3.1 BigFlow

Table 1 Network-level feature set used in the experiments throughout this work [18]. Types: Host-based (Host to All), Flow-based (Source to Destination, Destination to Source, Both)

Features:

- Number of Packets,
- Number of Bytes,
- Average Packet Size,
- Percentage of Packets (PSH Flag),
- Percentage of Packets (SYN and FIN Flags),
- Percentage of Packets (FIN Flag),
- Percentage of Packets (SYN Flag),
- Percentage of Packets (ACK Flag),
- Percentage of Packets (RST Flag),
- Percentage of Packets (ICMP Redirect Flag),
- Percentage of Packets (ICMP Redirect Flag),

- Percentage of Packets (ICMP Time Exceeded Flag),
- Percentage of Packets (ICMP Unreachable Flag),
- Percentage of Packets (ICMP Other Types Flag),
- Throughput in Bytes,
- Protocol

BigFlow destaca em sua secção 2 (background) o processamento de streams [18, 19], a preferencia de NIDS por anomalia em contraste aos NIDS por assinatura [30, 31, 32], a variabilidade e evolução dos padrões de tráfego em redes de propósito geral [9, 11, 20], a necessidade de atualização regular do modelo classificador [8, 9, 10, 20] e o tratamento de eventos onde a confiança resultante da classificação é baixa [9, 12, 13].

Também destaca em sua secção 3 (MAWIFlow) que datasets adequados para NIDS são poucos devido o conjunto de qualidades que os mesmos devem atender como realismo, validade, etiquetamento, grande variabilidade e reprodutividade (disponibilidade pública) [8, 9, 10, 17, 38].

Para avaliar o desempenho de NIDS o dataset MAWIFlow é proposto. Originário do 'Packet traces from WIDE backbone, samplepoint-F' composto por seções de captura de pacotes diárias de 15 minutos de um link de 1Gbps entre Japão e EUA, com inicio em 2006 continuamente até hoje, anonimizados [22], etiquetados por MAWILab [8]. Desse dataset original apenas os eventos de 2016 são utilizados e desses 158 atributos são extraídas resultando em 7.9 TB de captura de pacotes. Além disso, os dados são estratificados [24] para redução de seu tamanho a um centésimo mantendo as proporções de etiquetas (Ataque e Normal) facilitando o compartilhamento e avaliação de NIDS além de atender as qualidades anteriormente mencionadas.

Com o dataset MAWIFlow original e reduzido foram avaliados quatro classificadores [42, 43, 44, 45] da literatura em dois modos de operação quanto seus dados de treinamento (ambos contendo uma semana de captura) o primeiro usando somente a primeira semana do ano e as demais como teste e o segundo modo usando a semana anterior como treinamento e a seguinte como teste. Demonstrando, com 62 atributos, que a qualidade da classificação retrai com o tempo quando não há atualização frequente do modelo classificador.



# Capítulo 4

## IMPLEMENTAÇÃO E TESTES

---

### 4.1 Descrição da Implementação

- offline, online, ND, Clustering
- observação/Considerações de paralelização

Notas sobre implementação Python/Kafka/Minas (não escala como esperado)

Dificuldade no processamento distribuído em Flink.

- complexidade bigO (?)

### 4.2 Cenário de Teste

Para testar e demonstrar essa implementação um cenário de aplicação é construído onde seria vantajoso distribuir o processamento segundo o modelo *fog*. Alguns cenários de exemplo são casos onde deve-se tomar ação caso uma classe ou anomalia seja detectada

- detecção de intrusão - Arquitetura guilherme (dispositivos pequenos vs cloud) (CASSA-LES et al., 2019)

- BigFlow com dataset atual e maior dataset kdd99

### 4.3 Experimentos e Resultados

- gráficos, tempos, tabelas... - análises e comentários

Mostrar alguma implementação já feita e que esteja funcionando minimamente

Mostrar resultados mesmo que sejam bem simples e básicos, apenas para demonstrar que vc domina o ambiente e as ferramentas e que está apto a avançar no trabalho

passos feitos/a fazer 1. Entender Minas 2. Analisar/descrever dataset KDD 3. Notas sobre implementação Python/Kafka/Minas (não escala como esperado) 4. BigFlow (dataset mais novo, usa flink) 5. Plataforma Flink (processamento distribuído) Proposta 6. Implementar minas em Scala/Flink 7. Testar com datasets KDD e BigFlow 8. Validar/Comparar métricas com seus trabalhos correspondentes

- Descrição do hardware utilizado pode conter: - Arch, OS, Kernel, - CPU (core, thread, freq), - RAM (total/free size, freq), - Disk (total/free size, seq RW, rand RW), - Net IO between nodes (direct crossover, switched, wireless, to cloud) (bandwidth, latency). essas métricas permitem relacionar trade-offs para as questões de fog: Processar em node, edge ou cloud?

Provavelmente vou retirar o kafka da jogada em node/edge, deixando apenas em cloud.

# Capítulo 5

## CRONOGRAMA

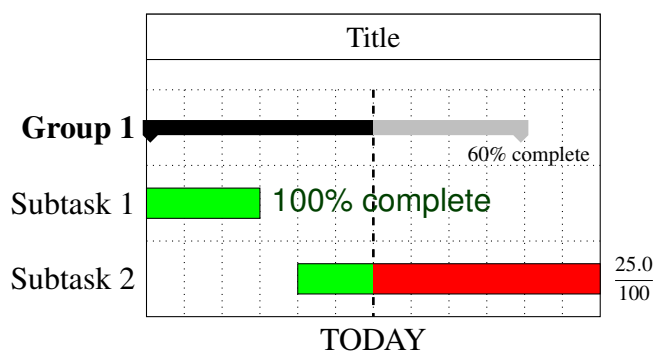
---

cap 5: Cronograma até a defesa

### 5.1 Cronograma

Lista de etapas

Quadro (linhas: etapas x colunas: meses)



Etapas previstas:

- E1 - Implementação distribuída do MINAS em uma rede de SBC com Rapberry: meses 01 a 03
- E2 - Preparação da base de dados para treinamento e teste: Mês 02
- E3 - Teste ... : meses 04 a 07
-

## REFERÊNCIAS

---

CASSALES, G. W.; SENGHER, H.; DE FARIA, E. R.; BIFET, A. IDSA-IoT: An Intrusion Detection System Architecture for IoT Networks. In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. [s.n.], 2019. p. 1–7. ISBN 978-1-7281-2999-0. ISSN 1530-1346. Disponível em: <https://ieeexplore.ieee.org/document/8969609/>.

GABER, M. M.; ZASLAVSKY, A.; KRISHNASWAMY, S. Mining data streams: A review. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 34, n. 2, p. 18–26, jun. 2005. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/1083784.1083789>.

GAMA, J.; RODRIGUES, P. P. Data stream processing. In: \_\_\_\_\_. *Learning from Data Streams: Processing Techniques in Sensor Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 25–39. ISBN 978-3-540-73679-0. Disponível em: [https://doi.org/10.1007/3-540-73679-4\\_3](https://doi.org/10.1007/3-540-73679-4_3).

KREPS, J. *Questioning the Lambda Architecture* – O'Reilly. 2014. 10 p. Disponível em: <https://www.oreilly.com/radar/questioning-the-lambda-architecture/>.

LOPEZ, M. A. *A monitoring and threat detection system using stream processing as a virtual function for Big Data*. Tese (Theses) — Sorbonne Université ; Universidade federal do Rio de Janeiro, jun. 2018. Disponível em: <https://tel.archives-ouvertes.fr/tel-02111017>.

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

MELL, P.; GRANCE, T. The NIST definition of cloud computing: Recommendations of the National Institute of Standards and Technology. In: NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Public Cloud Computing: Security and Privacy Guidelines*. 2012. p. 97–101. ISBN 9781620819821. Disponível em: <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>.

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, Institute of Electrical and Electronics Engineers Inc., v. 3, n. 5, p. 637–646, oct 2016. ISSN 23274662. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7488250>.