Detecção do Comportamento da Névoa em Sistemas IoT

Franklin M. Ribeiro Junior[1, 2], Reinaldo A. C. Bianchi[3], Carlos A. Kamienski[1]

[1] Universidade Federal do ABC (UFABC)
São Paulo, Brasil

[2] Instituto Federal do Maranhão (IFMA)
Maranhão, Brasil

[3] Centro Universitário FEI (FEI)
São Bernardo do Campo - SP, Brasil

franklin.junior@ufabc.edu.br, rbianchi@fei.edu.br,
carlos.kamienski@ufabc.edu.br

Abstract. *A fog-based IoT system has thousands of heterogeneous devices with distinct constraints. In this article, we propose a system that uses machine learning to group these devices' behaviors and identify each fog node's performance anomalies. We simulate different behaviors to evaluate our system, using the MeanShift, BIRCH, and K-Means clustering algorithms. We also evaluate the clustering data models using Silhouette, Davies-Bouldin, and Calinski-Harabasz indexes. We observe that the system identifies simulated behaviors with at least 99% accuracy, using the K-Means algorithm and the Calinski-Harabasz index in clustering validation.*

Resumo. *Um sistema IoT baseado em névoa contém milhares de dispositivos heterogêneos com suas próprias limitações. Este artigo propõe um sistema que utiliza aprendizado de máquina para agrupar os comportamentos desses dispositivos e identificar anomalias no desempenho dos diferentes nós de névoa. O sistema foi avaliado para diferentes comportamentos simulados, com os algoritmos MeanShift, BIRCH e K-Means. Também foram validados os agrupamentos gerados pelos índices de Silhouette, Davies-Bouldin e Calinski-Harabasz, a fim de obter o modelo de dados mais acurado. O sistema identificou os comportamentos simulados com pelo menos 99% de acurácia, usando o algoritmo K-Means e o índice de Calinski-Harabasz.*

1. Introdução

Um sistema de Internet das Coisas (*Internet of Things* - IoT) contém sensores e atuadores que transferem dados através da Internet e executam decisões já processadas por um servidor externo [Atzori et al. 2017]. Já um sistema de IoT baseado em névoa computacional pode analisar dados localmente na borda e reduzir a dependência do sistema por um servidor em nuvem [Yousefpour et al. 2019]. A névoa também garante a continuidade do fluxo de dados na IoT, mesmo sem suporte da nuvem [Atlam et al. 2018], [Junior e Kamienski 2021].

Devido a heterogeneidade e a diversidade dos dispositivos IoT quanto aos recursos de hardware, cada nó de névoa funciona melhor de acordo com suas demandas e capacidades computacionais [Atlam et al. 2018]. Por isso, cada dispositivo de névoa pode ter comportamentos esperados[1] distintos. Por exemplo, um nó X pode se comportar captando 200 pacotes por minuto, com uso de RAM e CPU em 20%, e um outro nó Y espera receber 3.000 pacotes por minuto, com uso de CPU em 70% e 85% de RAM. Consequentemente, determinados comportamentos que são apropriados para alguns dispositivos podem ser considerados anômalos para outros. Além disso, o comportamento esperado de um mesmo nó também pode ser diferente, a depender do contexto (temporal ou espacial).

Alguns trabalhos pré-classificam possíveis anomalias de rede, mas não cobrem comportamentos inexplorados que apenas ocorrem durante a execução do sistema [Cook et al. 2020]. Outros trabalhos usam aprendizado não supervisionado para detectar anomalias, mas não consideram que o comportamento esperado de um nó névoa pode ser mutável dependendo do contexto e do fluxo de dados de cada dispositivo [Al-amri et al. 2021]. Por isso, este artigo propõe um sistema denominado IoT-Beeps (IoT Behavior Perception System) que aprende todos os comportamentos de um nó de névoa e identifica o comportamento esperado de cada dispositivo individualmente.

O IoT-Beeps opera em um nó de névoa e recebe registros de entrada que formam o comportamento do dispositivo, são eles: o uso de CPU, uso de RAM, vazão (*throughput*), número de pacotes e tempo de atraso entre os sensores e a névoa (*one-way-delay*). Em linhas gerais, o IoT-Beeps roda um algoritmo de aprendizado de máquina não supervisionado e possui três fases: 1) a primeira fase treina um modelo com dados históricos para agrupar os comportamentos da névoa em *clusters*; 2) a segunda fase prevê a qual *cluster* o registro de entrada atual pertence e; 3) a terceira fase inclui os dados de entrada nos dados históricos, reiniciando a primeira fase. Portanto, o IoT-Beeps retreina os modelos iterativamente para aprender novos comportamentos.

O IoT-Beeps foi avaliado com os algoritmos de agrupamento K-Means [Jain 2010], BIRCH [Zhang et al. 1997] e MeanShift [Cheng 1995], com uso dos índices de validação: Silhouette, Davies-Bouldin e Calinski-Harabasz [Ünlü e Xanthopoulos 2019]. A avaliação ocorreu em quatro cenários: (i) foi investigado qual dos três algoritmos agrupou adequadamente o modelo de dados inicial, (ii) foi executado um comportamento inesperado, mas conhecido pelo modelo, (iii) foi utilizado um comportamento desconhecido e (iv) foram simulados dois comportamentos inexplorados. Nos experimentos foram medidos os três índices de validação, o tempo de treino e seleção do modelo de dados, o número de comportamentos encontrados e a acurácia do modelo selecionado. Os resultados constataram que quando o IoT-Beeps executa o K-Means, com o índice de Calinski-Harabasz, o modelo de dados selecionado possui uma acurácia de pelo menos 99%.

O restante do artigo está organizado da seguinte maneira: a Seção 2 descreve a fundamentação, a Seção 3 apresenta os trabalhos relacionados, a Seção 4 detalha o IoT-Beeps, a Seção 5 explana a metodologia, a Seção 6 mostra os resultados, a Seção 7 discute os resultados, e finalmente, a Seção 8 apresenta a conclusão e trabalhos futuros.

---

[1] Neste artigo o comportamento esperado é definido pelo comportamento com maior número de ocorrências durante a execução de um nó (em um determinado contexto).

2. Fundamentação

Nesta seção foram descritos os estágios de um sistema IoT baseado em névoa computacional, os algoritmos de aprendizado de máquina não supervisionado e os índices validação utilizados na pesquisa.

2.1. Estágios de IoT

Zyrianoff et al. (2019) definem um sistema de IoT baseado em névoa com quatro estágios (Figura 1): (i) o estágio de coisa contém atuadores para executar uma decisão e sensores para coletar os dados, (ii) o estágio de bruma é uma subdivisão da névoa [Linaje et al. 2019] e pré-processa os dados recebidos por sensores, (iii) o estágio de névoa pode armazenar e analisar dados mesmo durante uma desconexão da Internet e (iv) o estágio de nuvem computa e armazena dados históricos para aprimorar decisões futuras. Como os estágios de bruma e névoa possuem vários nós com diferentes recursos e comportamentos distintos, é impossível estabelecer um padrão para definir o comportamento esperado de cada nó.

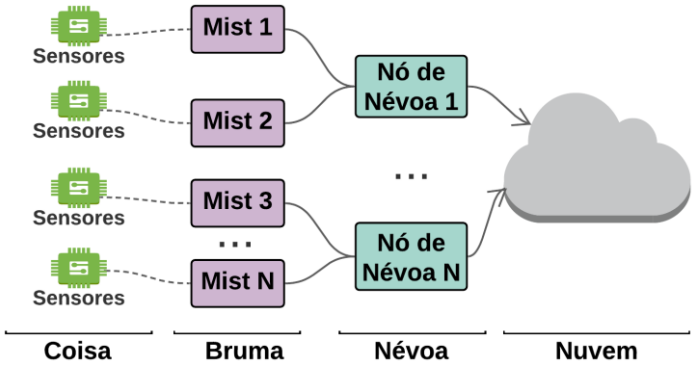

Figura 1. Estágios de IoT.

2.2. Aprendizado de máquina não supervisionado

Os algoritmos de aprendizado de máquina não supervisionado, também conhecidos como algoritmos de agrupamento (*clustering*), podem agrupar amostras de dados sem conhecimento prévio de nenhuma categoria de dados [Xu e Tian 2015]. Isso porque, esses algoritmos identificam similaridades entre as amostras para formar um grupo (*cluster*) de dados. Os algoritmos de agrupamento podem perceber grupos de dados, pelas semelhanças dos dados, usando diferentes estratégias como densidade dos dados ou distância entre amostras. Portanto, um algoritmo de *clustering* treina os dados para formar um modelo de dados que denota os agrupamentos encontrados.

O desempenho de um agrupamento de dados pode ser mensurado por índices de validação como: o coeficiente de Silhouette, Davies-Bouldin e Calinski-Harabasz [Ünlü e Xanthopoulos 2019]. O coeficiente de Silhouette indica que o modelo de dados agrupa melhor quando o valor do índice está mais próximo de um. O índice de Davies-Bouldin indica o melhor agrupamento quando o valor se aproxima de zero. Já o índice de Calinski-Harabasz mostra o melhor agrupamento, ao aferir o maior valor possível.

3. Trabalhos Relacionados

Cook et al. (2020) mencionam que a detecção de anomalias em IoT possui como desafios a percepção do contexto, o conhecimento prévio dos dados, a escalabilidade e

as restrições de recursos dos dispositivos IoT. Os autores também apontam que o uso de um classificador para detecção de anomalias pode ter dificuldade em identificar novas anomalias, que não foram treinadas pelo modelo de dados.

Power e Kotonya (2018) propuseram um arcabouço para detectar e prever falhas usando aprendizado de máquina, mas consideraram um cenário específico com algoritmos de aprendizado supervisionado e não se preocuparam em classificar diferentes comportamentos inesperados. Xu et al. (2019) apresentaram uma abordagem de aprendizado semi-supervisionado para detectar anomalias de rede, mas focaram apenas em ataques negação de serviço (DoS) e não em outros tipos de comportamentos.

Apostol et al. (2021) usaram aprendizado de máquina profundo não supervisionado para detectar anomalias, mas apenas consideraram ameaças causadas por ataques externos e não focaram em comportamentos inerentes ao fluxo de dados do próprio sistema. Nõmm e Bahşi (2018) utilizaram o aprendizado não supervisionado para detectar ataques em um sistema IoT, mas treinaram apenas um único (e imutável) modelo de dados para cada dispositivo IoT.

Al-amri et al. (2021) realizaram uma revisão de literatura e identificaram 23 trabalhos sobre detecção de anomalias em IoT com uso de aprendizado de máquina. Eles apontaram que todos os 17 estudos que utilizaram aprendizado não supervisionado trataram as anomalias somente como valores *outliers*, sem considerar o contexto.

Dentre os trabalhos relacionados, foi observado que os autores não consideraram que cada um dos milhares de nós de névoa pode ter seu próprio comportamento esperado, e que esse comportamento pode mudar de acordo com o contexto (como o período do dia, mês ou estação do ano). Por isso, a principal contribuição desse trabalho consiste em apresentar um sistema que é capaz de identificar iterativamente novos comportamentos de cada nó de névoa em *clusters* e de perceber o comportamento esperado de um nó, considerando o contexto temporal.

4. IoT-Beeps

O IoT-Beeps identifica comportamentos de um nó de névoa, através de um algoritmo de aprendizado de máquina não supervisionado e foi implementado em linguagem de programação Python, pois ela possui bibliotecas maduras de aprendizado de máquina, como o *scikit-learn*[2].

Antes de iniciar o IoT-Beeps é preciso executar um agente denominado de IoT Monitor que coleta os registros de um nó de névoa: como o uso de CPU (%), uso de RAM (%), vazão (Kb/s), atraso de pacotes recebidos (em milissegundos) e o número de pacotes recebidos. Porém, vale ressaltar que outras métricas de hardware também poderiam ser utilizadas para perceber o comportamento do nó de névoa. Como IoT-Beeps não tem conhecimento prévio de nenhum comportamento da névoa, é recomendável que antes de executá-lo, o sistema opere sem qualquer tráfego de pacotes para garantir que a base de dados inicial terá ao menos dois comportamentos antes da execução: (i) com o fluxo normal de dados e (ii) com a carga de pacotes igual a zero.

Para separar os registros coletados em um contexto temporal diário (por exemplo: manhã, tarde e noite), o sistema pode dividir as amostras para formar três

[2] https://scikit-learn.org/

bases de dados iniciais, uma para cada período do dia. Dessa forma, é possível aprender o comportamento esperado de um nó em momentos distintos.

Antes da inicialização, o sistema verifica o contexto temporal e uma das bases iniciais coletadas assume a nomenclatura de *FullDB*. Simultaneamente a execução do IoT-Beeps, o IoT Monitor alimenta uma base de dados chamada *SamplesDB* em um intervalo de *T* segundos (Figura 2).

Na inicialização (Figura 2), o IoT-Beeps executa o *laço 1* que sucede os seguintes passos: (i) lê a base de dados inicial nomeada como *FullDB*, (ii) treina diferentes modelos de dados, variando parâmetros como o número de agrupamentos, (iii) seleciona um dos modelos pelo cálculo do índice de validação e (iv) obtém um modelo de dados como saída e define o *cluster* dominante (que é o *cluster* do comportamento esperado). No *laço 2,* o IoT-Beeps (v) espera um período de tempo *T*, (vi) obtém as amostras do *SamplesDB*, (vii) prevê o comportamento da amostra e se ele é igual ao comportamento esperado (caso não seja, envia um alerta para a nuvem), (viii) o sistema também insere todos os dados de *SamplesDB* em *FullDB* e (ix) apaga os registros de *SamplesDB*. Quando ocorre uma mudança no contexto temporal, os dados de *FullDB* são preservados para serem reutilizados mais tarde, entretanto, uma outra base de dados inicial assume essa nomenclatura.

No IoT-Beeps, o *laço 2* sempre efetua a comparação dos comportamentos das amostras com o *cluster* dominante por um tempo $i * T$, para descobrir se o nó está se comportando como esperado. Ao final da execução do *laço 2*, o sistema retreina *FullDB* para gerar novos modelos que possivelmente contenham comportamentos inexplorados. No experimento foi utilizado o valor aproximado de $T = 7$ segundos e $i * T = 6$ minutos (ver Seção 5.5). Porém, os valores de i e T podem ser alterados e definidos pelo usuário.

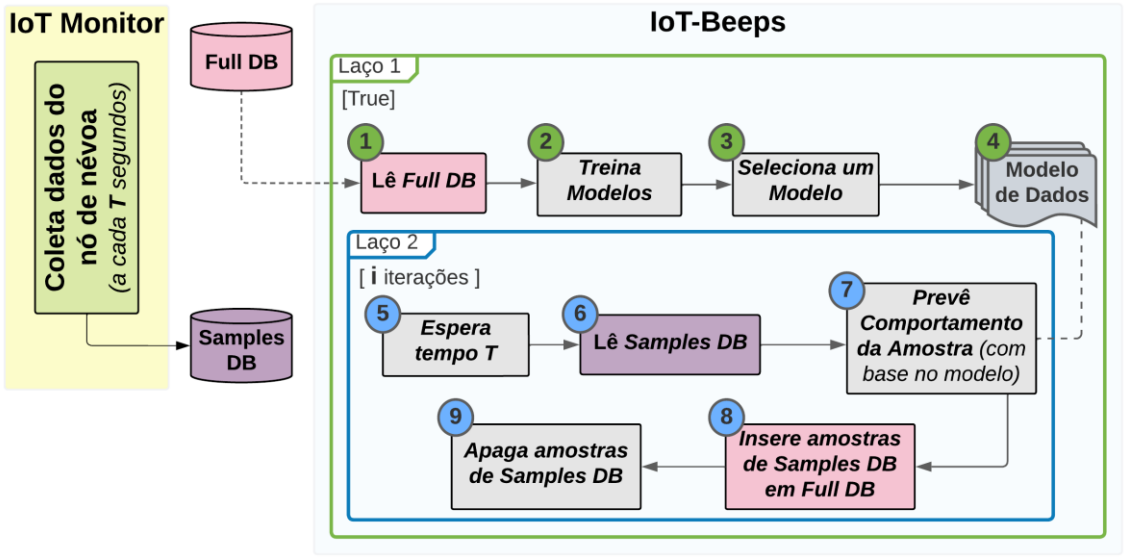

Figura 2. IoT Behavior Perception System (IoT-Beeps).

5. Metodologia

Esta seção descreve a geração da base de dados inicial, o ambiente experimental, as métricas avaliadas e os cenários que foram explorados na avaliação do IoT-Beeps. No experimento foi utilizado o simulador SenSE [Zyrianoff et al. 2017] para simular milhares de sensores com pacotes que contêm na carga útil, o tempo de criação do

pacote (*timestamp*). Foi utilizada uma ferramenta nativa do Linux, denominada de *stress-ng* para sobrecarregar o uso de CPU e de RAM do nó de névoa. Também foi utilizado um sistema de gerenciamento de banco de dados SQL denominado CrateDB[3], para suportar as bases de dados *FullDB* e *SamplesDB*.

5.1. Ambiente Experimental

O ambiente experimental (Figura 3) possui os quatro estágios de IoT (Seção 2.1), onde foram executados: (i) o simulador SenSE em correspondência aos estágios de coisa e bruma, (ii) o Mosquitto MQTT broker[4], a ferramenta *stress-ng*, o banco de dados CrateDB, o IoT-Monitor e o IoT-Beeps no estágio de névoa, e finalmente (iii) o Mosquitto MQTT broker no estágio de nuvem. O simulador SenSE, o nó de névoa e a nuvem foram executados em máquinas virtuais distintas, de processador Intel Xeon E5620, com 2 CPUs de 2.4 GHz e 4 GB de RAM, em um sistema Linux Ubuntu 18.04.

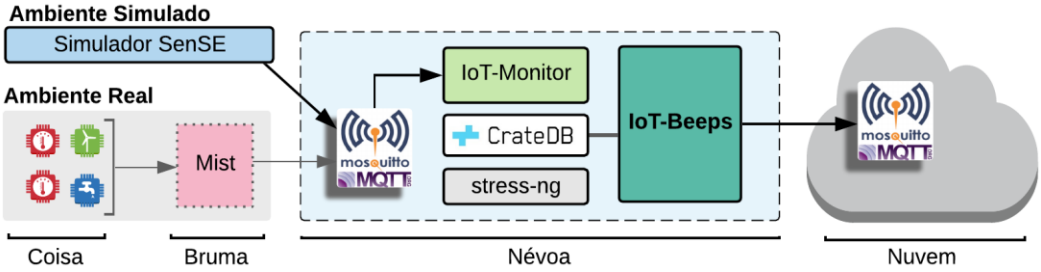

Figura 3. Ambiente Experimental.

5.2. Comportamentos Simulados para a Base de Dados Inicial

A base de dados inicial [Dataset 2022] foi obtida através de simulações do SenSE e da ferramenta *stress-ng*, onde foram simulados quatro comportamentos. Foi utilizado o IoT Monitor para coletar os dados da base de dados inicial (*Full_DB*), sendo: o uso de CPU, o uso de RAM, o atraso dos pacotes (do simulador SenSE à névoa), o número de pacotes e a vazão (*throughput*). Entretanto, outras métricas também poderiam ser coletadas pelo IoT Monitor para compor o comportamento de um nó. Os quatro comportamentos iniciais foram simulados da seguinte maneira:

- *Comportamento C0*: Carga baixa de pacotes (200 pacotes por minuto);
- *Comportamento C1*: Carga nula de pacotes (não houve tráfego, logo o número de pacotes transmitidos é igual a zero);
- *Comportamento C2*: Carga baixa de pacotes (200 pacotes por minuto) e sobrecarrega de CPU em 70%;
- *Comportamento C3*: Carga média de pacotes (500 pacotes por minuto) e sobrecarrega de CPU em 70%.

Na simulação, foi executado o comportamento C0 por duas horas e os demais (C1, C2 e C3) por 1:30 hora. Portanto, C0 foi considerado o comportamento dominante.

[3] https://crate.io/
[4] https://mosquitto.org/

5.3. Métricas

Na avaliação do IoT-Beeps foram analisadas as seguintes métricas:

- *Índice de Silhouette (SC)*, *Índice de Davies-Bouldin (DB)* e *Índice de Calinski-Harabasz (CH)*, apresentados na Seção 2.2;
- *Tempo de treino e seleção*: tempo para treinar, validar e selecionar apenas um dos modelos de dados (corresponde aos passos 2, 3 e 4 da Figura 2);
- *Matriz de confusão*: afere a acurácia do modelo de dados. Com ela é possível perceber se o IoT-Beeps identificou os comportamentos da névoa corretamente;
- *Comportamentos encontrados*: número de grupos encontrados no modelo após o cálculo dos índices de Silhouette, Davies-Bouldin e Calinski-Harabasz.

5.4. Algoritmos de Agrupamento

O IoT-Beeps gerou modelos de dados com 2 a 10 grupos para os algoritmos K-Means e BIRCH (variando o *threshold* do BIRCH com os valores de 0.05, 0.15 e 0.25). Para o BIRCH foram executados testes em duas configurações: BIRCH-1 e BIRCH-3. O BIRCH-1 possui *threshold* = 0.15, enquanto o BIRCH-3 calcula os três valores de *threshold* citados. Para o MeanShift, o IoT-Beeps gerou modelos variando a largura de banda (*bandwidth*) entre 0.1 e 0.6 (com intervalos de 0.05). Também foram calculados os índices de validação (Silhouette, Davies-Bouldin e Calinski-Harabasz) para determinar o modelo de dados que melhor agrupou os dados.

5.5. Cenários

Foram avaliados quatro cenários, onde o IoT-Beeps levou aproximadamente seis minutos para gerar um novo modelo e sete segundos para ler uma amostra. Foram executados os algoritmos BIRCH, K-Means e MeanShift em um primeiro cenário. Já os cenários subsequentes (2, 3 e 4) foram executados durante uma hora, onde apenas foi avaliado o algoritmo que identificou estritamente os quatro comportamentos simulados (ver Seção 5.2). Na avaliação, todos os cenários utilizaram a base de dados inicial [Dataset 2022] como ponto de partida. Os cenários avaliados foram:

- Cenário 1: Treinamento do modelo de dados inicial usando os algoritmos K-Means, MeanShift e BIRCH;
- Cenário 2: Foi simulado o comportamento dominante C0 por 58 minutos e em seguida foi executado o comportamento C1, por 2 minutos;
- Cenário 3: Foi simulado o comportamento dominante C0 por 40 minutos e em seguida, um comportamento desconhecido denominado C4. Para simular o comportamento C4, foi executada uma carga alta de 1.200 pacotes por minuto. Contudo, após dois minutos, o comportamento C0 permaneceu até o final do experimento;
- Cenário 4: Foi simulado o comportamento dominante C0, durante 22 minutos. Depois disso, um comportamento desconhecido C5 sobrecarregou a CPU do nó de névoa em 100% e a RAM em 75% durante 2 minutos. Em seguida, C0 executa entre os minutos 24 e 40, então é simulado um novo comportamento C4 (com carga de 1.200 pacotes por minuto), durante 2 minutos. Finalmente, C0 executa do minuto 42 até o final do experimento.

6. Resultados

Esta seção descreve os resultados obtidos para os cenários 1, 2, 3 e 4. Os dados coletados estão disponíveis em [Dataset 2022]. A métrica referente ao tempo de treino e seleção foi resultado de uma média de 30 experimentos replicados, onde também foram calculados os intervalos de confiança (margem de erro), com um nível de confiança de 95%.

6.1. Cenário 1: Treinamento do Modelo Inicial

Nesse cenário foi avaliado o IoT-Beeps com os algoritmos de agrupamento BIRCH, K-Means e MeanShift. Com o experimento, percebeu-se que o tempo para treinar e selecionar um modelo com o algoritmo MeanShift é 45.51 vezes (168.4/ 3.7) maior que no K-Means e 29.54 vezes (168.4/ 5.7) maior que no BIRCH-3, quando se usa o cálculo do índice de Calinski-Harabasz (Figura 4). Também foi observado que o tempo de seleção do modelo aferiu um valor maior para o índice de Silhouette, em contraste aos outros dois índices calculados.

Foi percebido que o BIRCH-1 apresentou resultados para o tempo de treino e seleção menores que o K-Means, enquanto que o BIRCH-3 reportou valores maiores que o K-Means. Porém, o uso do BIRCH-1 somente é viável se o IoT-Beeps tiver conhecimento prévio de qual valor de *threshold* gera o modelo mais apropriado.

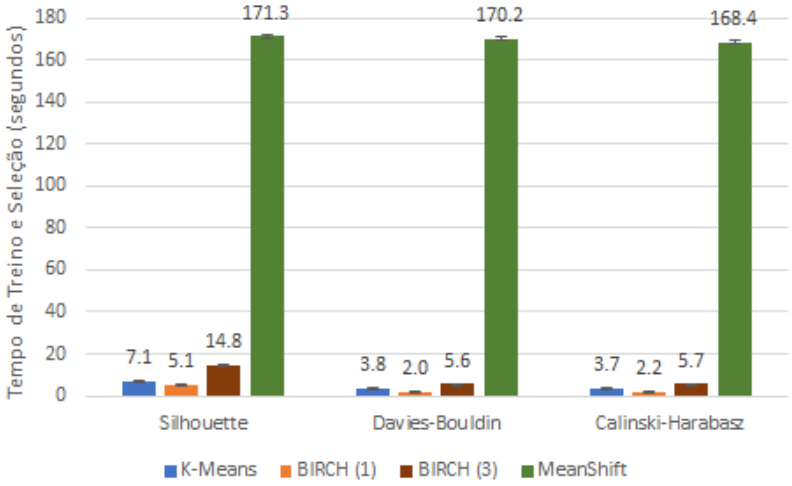

Figura 4. Tempo de Treino e Seleção.

Após treinar os modelos e observar os índices calculados nas Figuras 5, 6 e 7, foi percebido que o algoritmo K-Means gerou um modelo em que os dados ficaram mais bem agrupados com quatro grupos. Também foi possível inferir que o modelo selecionado pelo K-Means com quatro grupos, foi o mais apropriado ao observarmos a matriz de confusão desses grupos (Figura 8), já que ela apresentou a diagonal principal tendendo a aproximadamente 100% de acurácia (numa escala de 0.0 a 1.0) na relação entre linhas e colunas (que simbolizam os comportamentos simulados e os *clusters* previstos).

Foi percebido que o cálculo dos índices de validação para os modelos gerados pelo BIRCH e pelo MeanShift, não obtiveram os quatro grupos referentes aos quatro comportamentos simulados. Contudo, foi constatado que o uso do K-Means no IoT-Beeps gerou um modelo de dados com os quatro grupos que correspondem aos mesmos

quatro comportamentos simulados (ver Seção 5.2 e Figura 8). Por isso, somente o algoritmo K-Means foi avaliado nos cenários subsequentes.

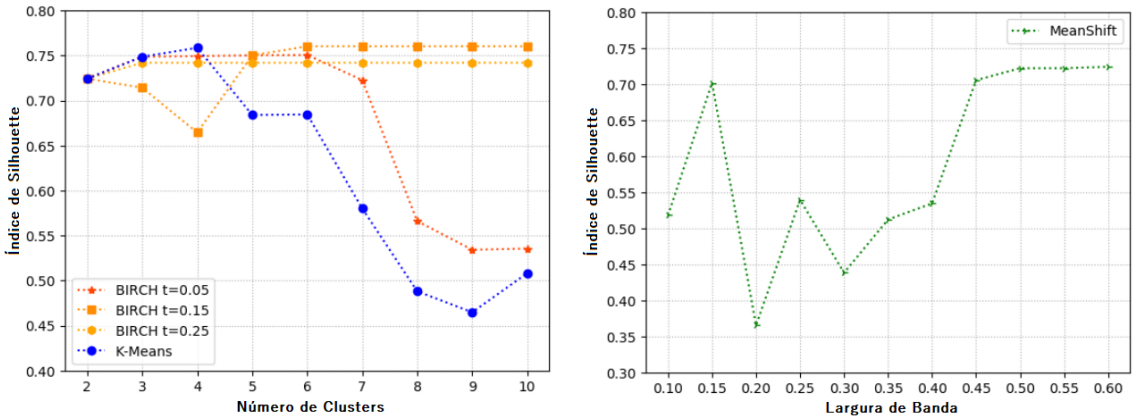

Figura 5. Valor do índice de Silhouette em relação aos grupos encontrados.

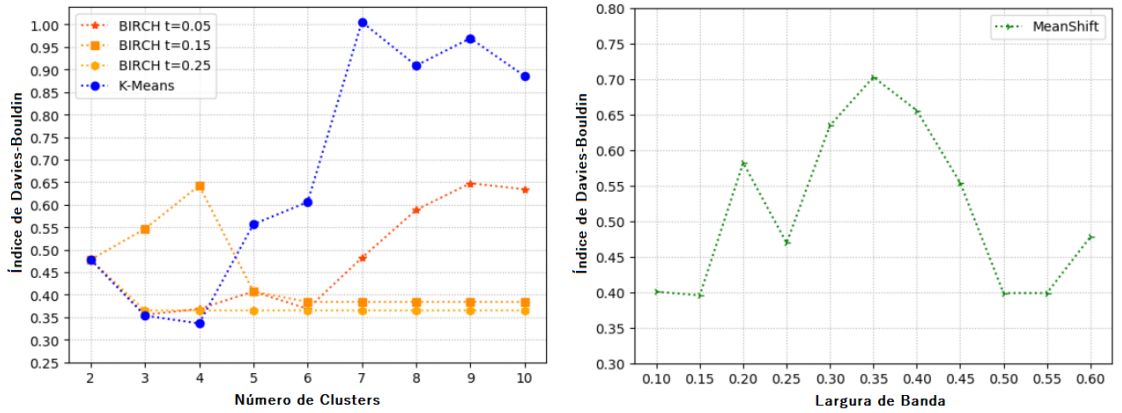

Figura 6. Valor do índice de Davies-Bouldin em relação aos grupos encontrados.

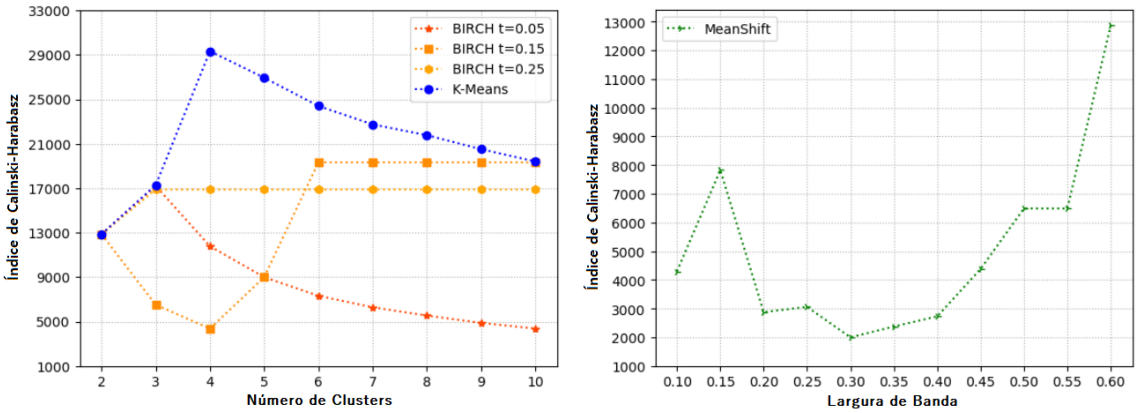

Figura 7. Valor do índice de Calinski-Harabasz em relação aos grupos encontrados.

6.2. Cenário 2: Simulação de comportamentos conhecidos pelo modelo

No cenário 2, foi simulado um experimento de uma hora, onde o IoT-Beeps gera um novo modelo de dados a cada rodada de seis minutos. No experimento, o comportamento C0 foi simulado durante 58 minutos de execução e após esse período foi executado C1, previamente conhecido pelo modelo de dados (Seção 5.2).

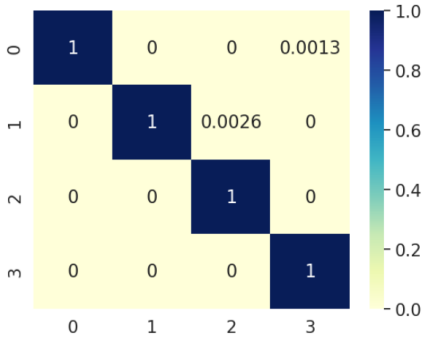

Figura 8. Matriz de Confusão dos grupos encontrados no modelo de dados inicial.

Na investigação foi observado que quando o IoT-Beeps usa o índice de Calinski-Harabasz (CH) para validar os modelos de dados, ele continua acertando os quatro comportamentos simulados (Tabela 1), com uma acurácia de pelo menos 99% (Figura 9). Também ficou evidente que ao calcular o número de comportamentos para os índices de Silhouette (SC) e Davies-Bouldin (DB), o sistema não identificou corretamente os quatro comportamentos simulados após 18 minutos de experimento (Tabela 1). Na Tabela 1, os índices SC e DB apenas identificaram três comportamentos no decorrer do experimento, enquanto o índice de Calinski-Harabasz permaneceu identificando os quatro comportamentos simulados até a última rodada.

Tabela 1. Grupos Encontrados no Cenário 2.

Rodada		-	1	2	3	4	5	6	7	8	9	10
Tempo (minutos)		0	6	12	18	24	30	36	42	48	54	60
Número de grupos	**SC**	4	4	3	3	3	3	3	3	3	3	3
	DB	4	4	4	3	3	3	3	3	3	3	3
	CH	4	4	4	4	4	4	4	4	4	4	4

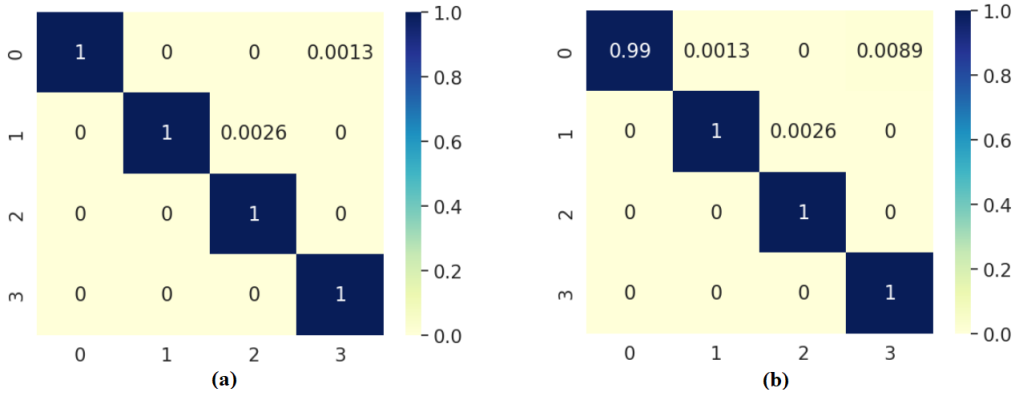

Figura 9. Matriz de confusão do cenário 2, para os (a) grupos encontrados no modelo inicial e (b) após uma hora de simulação.

6.3. Cenário 3: Simulação de um comportamento desconhecido

No cenário 3 foram simuladas 10 rodadas (de seis minutos cada), onde o comportamento C0 foi simulado nas seis primeiras e na rodada 7 foi executado o comportamento C4 (Seção 5.5), desconhecido pelo modelo de dados. Na rodada 7 foi observado que o índice de Calinski-Harabasz calculou o modelo de dados corretamente, antes e depois do novo comportamento C4, já que ele encontrou quatro *clusters* antes da rodada 7 e cinco grupos a partir da rodada 7 (ver Tabela 2 e Figura 10).

Com a matriz de confusão, foi percebido que os grupos previstos coincidem com os comportamentos simulados C0, C1, C2, C3 (Seção 5.2) e C4 (Seção 5.5), em pelo menos 99% de correspondência (ver Figura 10). Também foi constatado que o índice de Silhouette (SC) não calculou adequadamente o número de grupos após a rodada 2 e que o mesmo problema foi observado para o índice Davies-Bouldin (DB) na rodada 7.

Tabela 2. Grupos encontrados no Cenário 3.

Rodadas		-	1	2	3	4	5	6	7	8	9	10
Tempo (minutos)		**0**	**6**	**12**	**18**	**24**	**30**	**36**	**42**	**48**	**54**	**60**
Número de grupos	**SC**	4	4	4	3	3	3	3	2	2	2	2
	DB	4	4	4	4	4	4	4	2	2	2	2
	CH	4	4	4	4	4	4	4	5	5	5	5

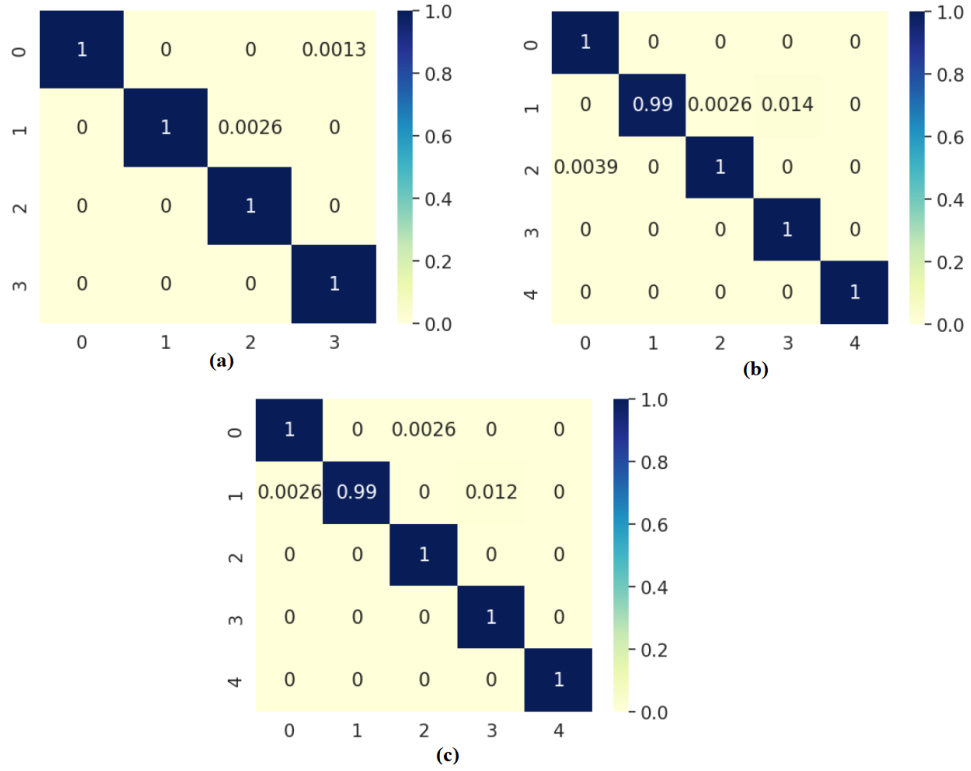

Figura 10. Matriz de confusão do cenário 3, para (a) os grupos encontrados com o modelo inicial, (b) após 42 minutos de execução, e depois de (c) uma hora de simulação.

6.4. Cenário 4: Simulação de dois comportamentos desconhecidos

No cenário 4 foi simulado um experimento de uma hora, com 10 rodadas de seis minutos, onde o IoT-Beeps gerou um novo modelo de dados a cada rodada (Tabela 3). Durante o experimento foram simulados dois comportamentos desconhecidos pelo modelo, o primeiro na rodada 4 e o segundo na rodada 7 (Seção 5.5).

Tabela 3. Grupos encontrados no Cenário 4.

Rodadas		-	1	2	3	4	5	6	7	8	9	10
Tempo (Minutos)		**0**	**6**	**12**	**18**	**24**	**30**	**36**	**42**	**48**	**54**	**60**
Número de grupos	**SC**	4	4	4	4	5	5	5	2	2	2	2
	DB	4	4	4	4	5	5	5	2	2	2	2
	CH	4	4	4	4	5	5	5	7	7	7	7

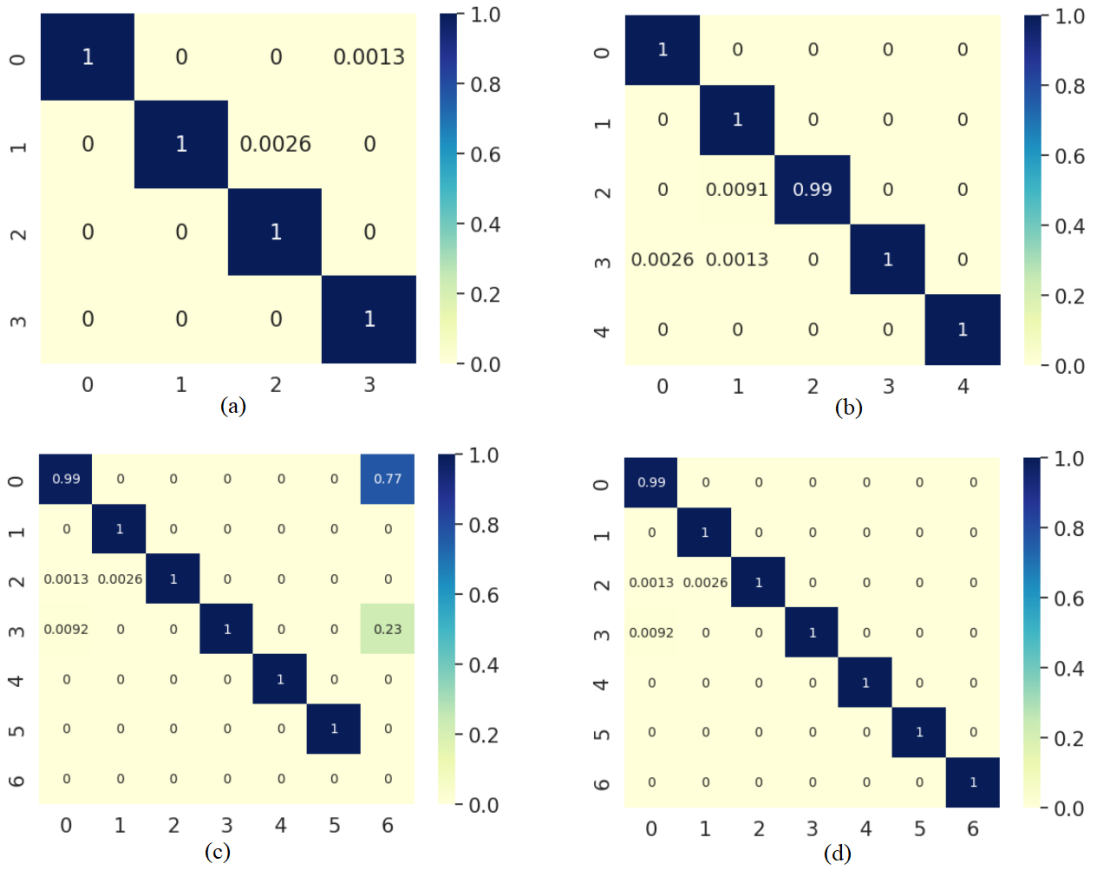

Figura 11. Matriz de confusão do cenário 4 (a) para os grupos encontrados no modelo inicial, (b) depois de 24 minutos de execução, (c) após 42 minutos de simulação (considerando apenas 6 comportamentos) e (d) após 42 minutos de simulação com os 7 comportamentos encontrados.

Foi observado que todos os índices de validação identificaram corretamente os quatro comportamentos até a rodada 3, bem como o novo comportamento a partir da rodada 4. No entanto, os índices de SC e DB somente identificaram dois comportamentos na rodada 7. Também foi notado que na rodada 7 o índice de Calinski-Harabasz encontrou sete comportamentos, ao invés de seis (Tabela 3).

Na rodada 7, mesmo se considerarmos apenas os seis comportamentos simulados, é possível observar que o IoT-Beeps agrupou os comportamentos C0, C1, C2, C3, C4 e C5 de maneira adequada (Figura 11c). No entanto, o índice de Calinski-Harabasz encontrou sete comportamentos. Ao analisar os dados da rodada 7 em [Dataset 2022], é possível perceber que esse sétimo comportamento agrupa apenas 22 amostras e que elas foram agrupadas corretamente, já que elas tem um valor de atraso de pacotes de pelo menos 150 vezes maior que o valor de atraso das outras 3.668 amostras (ver *https://github.com/FranklinMRJ/cenarios_iot/blob/main/README.md* e [Dataset 2022]). Portanto, a partir da rodada 7, a validação do modelo com o índice Calinski-Harabasz mostrou dois novos comportamentos simulados corretamente e mais um comportamento oculto, que possui amostras com um atraso muito acima do esperado (Figura 11d e Tabela 3). Além disso, os dados do sétimo grupo também poderiam ser pré-processados e tratados como *outliers*, já que eles não aparecem de maneira sequencial na base de dados.

7. Discussão

Com os resultados foi possível perceber que o IoT-Beeps, com o algoritmo MeanShift levou cerca de 170 segundos para treinar e selecionar um modelo de dados, sendo que esse tempo pode representar muito para sistemas IoT que exigem uma resposta mais rápida. Foi observado que o BIRCH apresentou um tempo de treinamento semelhante ao K-Means, porém o IoT-Beeps não identificou corretamente os quatro comportamentos simulados, mesmo alterando o valor do *threshold* do BIRCH.

Nos cenários avaliados, foi constatado que o IoT-Beeps identificou pelo menos 99% dos comportamentos simulados corretamente. No entanto, essa taxa de acerto apenas se manteve quando o modelo de dados foi gerado pelo algoritmo K-Means, com a validação mensurada pelo índice de Calinski-Harabasz.

Com os resultados, foi observado que o IoT-Beeps pode gerar novos modelos que percebem comportamentos desconhecidos em um sistema IoT. Portanto, pode-se inferir que o IoT-Beeps não identifica apenas anomalias na névoa, mas também quaisquer tipos de comportamentos dos nós de névoa.

Diferentemente dos trabalhos que utilizam aprendizado de máquina não supervisionado para detectar anomalias [Apostol et al. 2021], [Nõmm e Bahşi 2018], [Al-amri et al. 2021], o IoT-Beeps foca em aprender todos os comportamentos de um nó de névoa de maneira genérica e iterativa. Além disso, o IoT-Beeps pode considerar o contexto para perceber o comportamento esperado de um nó de névoa.

Em uma cidade inteligente, sensores de luminosidade poderiam enviar pacotes a cada uma hora durante o período da manhã. Eles também poderiam enviar dados a noite, mas em um intervalo de 1 minuto. Logo, num dia de tempestade, em que esses sensores passem a coletar dados mais frequentemente, o IoT-Beeps poderia detectar que a névoa está se comportando de maneira errada. O mesmo poderia acontecer em um sistema de irrigação inteligente, onde a névoa pode demandar mais ou menos recursos de hardware para analisar os dados, já que a frequência da análise de dados nesse tipo de aplicação depende de outros fatores como a umidade do solo, o tipo de planta ou a estação do ano.

8. Conclusão

Este artigo propõe e avalia um sistema denominado IoT-Beeps, para detectar o comportamento da névoa em um sistema IoT. A solução usa aprendizagem de máquina não supervisionada para agrupar os comportamentos em *clusters* e identificar os comportamentos de cada nó de névoa, sem conhecimento prévio dos dados. O IoT-Beeps foi avaliado com três algoritmos de agrupamento (K-Means, MeanShift e BIRCH), enquanto os modelos de dados foram validados com os índices de Silhouette, Calinski-Harabasz e Davies-Bouldin. Foi observado que o IoT-Beeps identificou diferentes comportamentos da névoa com pelo menos 99% de acurácia, quando utilizado o K-Means com a validação pelo índice de Calinski-Harabasz.

Com o IoT-Beeps é possível avançar na detecção de anomalias relacionadas ao fluxo de dados e a saúde dos dispositivos da névoa, pois ele mapeia todos os comportamentos de cada um dos nós de névoa e pode treinar bases de dados distintas para diferentes contextos. Como trabalho futuro espera-se avaliar a detecção de mais de um comportamento dominante, considerando diferentes contextos e aplicações.

Referências

Al-amri R., Murugesan R.K., Man M., Abdulateef A.F., Al-Sharafi M.A., Alkahtani A.A. (2021) “A Review of Machine Learning and Deep Learning Techniques for Anomaly Detection in IoT Data”. Applied Sciences. 11(12), 5320.

Apostol I., Preda M., Nila C., and Bica I., (2021) “IoT Botnet Anomaly Detection Using Unsupervised Deep Learning,” Electronics, vol. 10, no. 16.

Atlam H., Walters R., and Wills G. (2018) “Fog Computing and the Internet of Things: A Review,” Big Data and Cognitive Computing, vol. 2, no. 2, p. 10.

Atzori, L., Iera, A. and Morabito, G. (2017) “Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm”. Ad Hoc Networks, 56, pp.122-140.

Cheng Y. (1995) “Mean shift, mode seeking, and clustering”, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, no. 8, pp. 790-799.

Cook A. A., Mısırlı G., and Fan Z. (2020) “Anomaly Detection for IoT Time-Series Data: A Survey”, in IEEE Internet of Things Journal, vol. 7, no. 7, pp. 6481-6494.

[Dataset] Dados do IoT Monitor (2022), *https://github.com/FranklinMRJ/cenarios_iot*.

Jain A. K., (2010) “Data clustering: 50 years beyond K-means”, Pattern Recognition Letters, Volume 31, Issue 8.

Junior F. M. R., Kamienski C. A. (2021) “A Survey on Trustworthiness for the Internet of Things”, in IEEE Access, vol. 9, pp. 42493-42514.

Linaje M., Berrocal J., Galan-Benitez A. (2019) “Mist and Edge Storage: Fair Storage Distribution in Sensor Networks”, in *IEEE Access*, vol. 7, pp. 123860-123876.

Nõmm S., and Bahşi H. (2018) “Unsupervised Anomaly Based Botnet Detection in IoT Networks”, 17th IEEE ICMLA, pp. 1048-1053, doi: 10.1109/ICMLA.2018.00171.

Power A., and Kotonya G. (2018) “A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems”, IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM).

Ünlü R., Xanthopoulos P. (2019) “Estimating the number of clusters in a dataset via consensus clustering”, Expert Systems with Applications.

Xu, D., Tian, Y. (2015) “A Comprehensive Survey of Clustering Algorithms”. Ann. Data. Sci. 2, 165–193. https://doi.org/10.1007/s40745-015-0040-1.

Xu S., Qian Y., and Hu R. Q. (2019) “A Semi-Supervised Learning Approach for Network Anomaly Detection in Fog Computing”, IEEE ICC, pp. 1-6.

Yousefpour A., Fung C., Nguyen T., Kadiyala K., Jalali F., Niakanlahiji A., Kong J., Jue J.P. (2019) “All one needs to know about fog computing and related edge computing paradigms: A complete survey”, Journal of Systems Architecture.

Zhang T., Ramakrishnan R., Livny M. (1997) “BIRCH: A New Data Clustering Algorithm and Its Applications”. Data Mining and Knowledge Discovery 1.

Zyrianoff, I. D. R., Borelli F., Kamienski C. (2017) “SenSE? Sensor Simulation Environment: Uma ferramenta para geração de tráfego IoT em larga escala”. Simpósio Brasileiro de Redes e Sistemas Distribuídos (SBRC).

Zyrianoff I., Heideker A., Silva D., Kleinschmidt J., Soininen J.-P., Cinotti S.T., and Kamienski C. (2019) “Architecting and Deploying IoT Smart Applications: A Performance–Oriented Approach,” Sensors, vol. 20, no. 1, p. 84.