Resiliência de Dados da Bruma Computacional na Internet das Coisas

Franklin Magalhães Ribeiro Junior[1,2], Carlos Alberto Kamienski[1]

[1]Universidade Federal do ABC (UFABC)
Brasil

[2]Insituto Federal do Maranhão (IFMA)
Brasil

{franklin.junior,cak}@ufabc.edu.br

Abstract: *A mist- and fog-based IoT system must resist to fog disconnections and interruptions because the data send by the mist may be lost. This paper introduces and evaluates the ReMITS solution, which stores data in the mist even during network disconnections. We also propose a compression algorithm called LoRa-SAX to reduce data delivery delay when connection returns. We evaluate ReMITS with a simulated workload of 5,000 sensors and 7 configurations for 1-, 5-, and 30-minute disconnection times. We observe that ReMITS delivered all packets to the fog and the LoRa-SAX algorithm combined with bzip2 reduced the packets arrival time by up to 98.5% and the data size by up to 93.6%.*

Resumo: *Um sistema IoT baseado em bruma (mist) e névoa computacional deve resistir a desconexões e a interrupções da névoa, já que os dados transmitidos pela bruma podem ser perdidos. Esse artigo propõe uma solução intitulada ReMITS, que persiste os dados na bruma, mesmo durante desconexões de rede. Também foi proposto um algoritmo de compressão (LoRa-SAX), para reduzir o atraso dos dados quando a conexão é retomada. O ReMITS foi avaliado com uma carga simulada de 5.000 sensores e com sete configurações, para 1, 5 e 30 minutos de desconexão. Foi observado que o ReMITS entregou todos os pacotes à névoa e que o LoRa-SAX combinado ao algoritmo bzip2 reduziu o tempo de chegada dos pacotes em até 98,5% e o tamanho dos dados em até 93,6%.*

1. Introdução

A Internet das coisas (IoT) é composta por milhares de dispositivos conectados em rede, que percebem um contexto e atuam sobre um determinado ambiente [Atzori et al., 2010]. A computação em bruma[1] (*mist computing*) compõe um sistema IoT baseado em névoa computacional. Por isso, ela pode ser definida como um nível (de hardware e software) da computação em névoa mais próximo dos sensores, onde geralmente desempenha o papel de um *gateway* que recebe dados dos sensores e por isso, possui recursos computacionais mais limitados que a névoa [Asif-Ur-Rahman et al., 2019].

Sistemas IoT baseados em bruma e névoa computacional devem lidar com dados de milhares de sensores e considerar restrições de rede, memória, armazenamento e processamento dos dispositivos [Atlam et al., 2018]. Por isso, é um desafio prover

[1] No artigo foi adotado o termo "computação em bruma" como tradução para *mist computing*, já que não há uma palavra em português amplamente aceita pela comunidade acadêmica, para representar esse conceito.

técnicas para manter a resiliência dos dados de um sistema IoT, numa eventual desconexão ou interrupção do serviço da névoa [Yousefpour et al., 2019]. O arcabouço TW-IoT (*Trustworthiness for IoT*) [Junior e Kamienski, 2021] propõe mecanismos para manter a resiliência de dados em um sistema IoT baseado em bruma e névoa computacional. Contudo, os mecanismos de resiliência e redução de dados na bruma (do TW-IoT) ainda não foram implementados ou avaliados.

Esse artigo propõe uma solução intitulada ReMITS (***Re**silient **M**ist-based **IoT** **S**olution*), baseada no sistema FIRST (*Fog-based IoT Resilient System*) [Ribeiro Junior e Kamienski, 2020] e que provê resiliência e redução no tamanho dos dados em IoT. Mesmo com a interrupção do serviço da névoa ou em uma desconexão de rede entre a bruma e a névoa, o ReMITS armazena os dados enviados pelos sensores na bruma. Quando a comunicação da bruma com a névoa retorna, o ReMITS reduz o tamanho dos dados armazenados, para diminuir o tempo de chegada dos dados a névoa, evitando inclusive o aumento no tempo de chegada de futuros pacotes.

A redução dos dados pode ser obtida por meio de técnicas de agregação, filtragem ou compactação. Porém, a compactação mantém os dados originais e pode reduzir o tamanho dos dados sem comprometer a confidencialidade da carga útil (*payload*) dos pacotes, já que os algoritmos de compressão de dados não precisam decriptar os pacotes para manipulá-los. Por isso, também é proposto um algoritmo intitulado LoRa-SAX, que comprime os pacotes recebidos pela bruma, mesmo com a carga útil cifrada.

O ReMITS foi avaliado num cenário análogo ao de uma fazenda inteligente [Kamienski et al., 2019], com uma carga simulada de 5.000 sensores. Foram avaliadas as métricas de tempo de chegada dos dados na névoa, tamanho dos dados, tempo de compressão dos dados na bruma, tempo de descompressão dos dados na névoa, uso de RAM e CPU em situações de 1, 5 e 30 minutos de desconexão de rede entre a bruma e a névoa, seguidas pela retomada da conexão. A avaliação também variou técnicas de redução de dados, com foco no algoritmo proposto LoRa-SAX combinado ao bzip2[2], contra os algoritmos de compressão bzip2, Zstandard[3], gzip[4], zlib[5] e lzma[6], além do uso do ReMITS sem nenhuma técnica de compactação de dados.

Os resultados obtidos apontaram que o ReMITS utilizou, em média, entre 2% a 2,5% de CPU e 0,4% de RAM da bruma. Além disso, o uso do LoRa-SAX combinado ao algoritmo bzip2 apresentou uma redução de até 93,6% no tamanho dos dados e de até 98,5% no tempo de chegada. Contudo, a configuração que utilizou o algoritmo Zstandard apresentou o menor tempo de compressão e descompressão dos dados.

O restante do trabalho está organizado da seguinte maneira: a Seção 2 relata a fundamentação, a Seção 3 menciona os trabalhos relacionados, a Seção 4 apresenta a solução ReMITS e o algoritmo proposto LoRa-SAX, a Seção 5 discorre a metodologia, a Seção 6 reporta os resultados, a Seção 7 discute os resultados e, finalmente, a Seção 8 comenta as conclusões e os trabalhos futuros.

2. Fundamentação

Nesta seção é abordada a compactação dos dados na IoT e os estágios arquiteturais de um sistema IoT baseado em bruma e névoa computacional.

[2] https://docs.python.org/3/library/bz2.html
[3] https://facebook.github.io/zstd/
[4] https://www.gzip.org/
[5] https://zlib.net/
[6] https://docs.python.org/3/library/lzma.html

2.1. Estágios de um Sistema IoT baseado em Bruma e Névoa Computacional

Um sistema IoT baseado em bruma e névoa computacional possui vantagens de escalabilidade, já que a bruma aumenta a autonomia dos dispositivos próximos da borda [Preden et al., 2015]. A arquitetura desse tipo de sistema pode ser dividida em quatro estágios [Zyrianoff et al., 2019], são eles: Coisa, Bruma, Névoa e Nuvem (Figura 1).

O estágio Coisa contém sensores e atuadores que coletam dados e atuam em um determinado ambiente [Yousefpour et al., 2019]. O estágio Bruma é conectado ao estágio Coisa e é considerado um nível de névoa [Linaje et al., 2019], [Zyrianoff et al., 2019]. Por isso, assim como a névoa, a bruma pode processar e armazenar dados, mas de forma limitada [Asif-Ur-Rahman et al., 2019]. O estágio Névoa lida com uma grande quantidade de dados e pode analisar e armazenar dados localmente, independentemente da conexão com a nuvem pela Internet. Já o estágio Nuvem fornece recursos de hardware virtualizados com grande capacidade computacional para armazenar e analisar grandes volumes de dados [Armbrust et al., 2010].

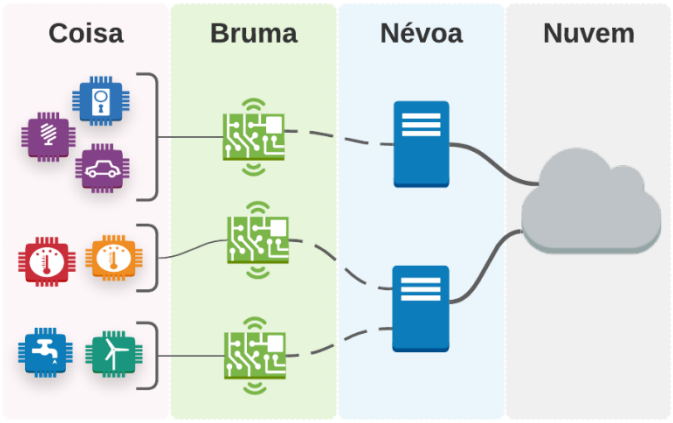

Figura 1. Estágios de um Sistema IoT baseado em Bruma e Névoa Computacional.

2.2. Redução de Dados na IoT

A redução no volume de dados gera impactos positivos no caminho entre os estágios de IoT [Gia et al., 2015], [Shi et al., 2015], [Negash et al., 2018], [Spiegel et al., 2018], [Azar et al., 2019], [Ribeiro et al., 2020], [Routray et al., 2020]. Em um sistema de saúde IoT, um nó de névoa pode usar um mecanismo de filtragem de dados para extrair informações relevantes de ECG, reduzindo o tamanho dos dados e a latência [Gia et al., 2015], [Azar et al., 2019]. Já a técnica de agregação de dados evita a transmissão de dados redundantes ou desnecessários [Shi et al., 2015], [Ribeiro et al., 2020].

Na bruma computacional, a redução dos dados pelas técnicas de filtragem e agregação gera perda nos detalhes das informações [Routray et al., 2020], comprometendo futuras decisões do sistema [Negash et al., 2018]. Além disso, caso os dados cheguem cifrados na bruma, as técnicas de filtragem e agregação precisam decriptar a carga útil dos pacotes para manipular os dados, expondo-os a invasores. Já os algoritmos de compressão de dados (gzip, Zstandard, zlib, bzip2 e lzma) reduzem o tamanho dos dados na IoT, sem arriscar a confidencialidade dos pacotes e sem causar perda de informações.

3. Trabalhos Relacionados

Cho et al. (2019) definem a resiliência como um requisito de *trustworthiness* em sistemas computacionais. Existem trabalhos sobre a resiliência na computação em borda e em

névoa na IoT [Jonathan et al., 2017], [Harchol et al., 2018], [Jeong et al., 2017] e [Al-Khafajiy et al., 2018]. Alguns autores abordaram a replicação de dados de um nó de névoa para outros nós, o que demanda mais recursos de computação do sistema [Jonathan et al., 2017], [Harchol et al., 2018]. Al-Khafajiy et al. (2018) propuseram um mecanismo de balanceamento de carga entre os nós de névoa para lidar com uma grande quantidade de dados recebidos pelos sensores, sendo que Jeong et al. (2017) consideraram mecanismos de autocura (*self-healing*) em caso de falha de um nó de névoa, alocando um novo nó concomitante à redundância ou migração dos dados. No entanto, essas investigações não introduziram mecanismos para detectar e recuperar falhas de conexão de rede entre os estágios de bruma e névoa em sistemas IoT.

Sistemas com uma rede tolerante a atrasos ou interrupções (DTN – *Delay Tolerant Network*) podem armazenar dados por um determinado período de tempo durante uma desconexão e enviá-los quando a conexão estiver disponível novamente [Fall, 2003]. Alguns trabalhos investigaram DTN no escopo de IoT [Luzuriaga, et al. 2017] [Kulatunga et al., 2017] [Castellano et al., 2018], mas apenas avaliando o atraso ou a perda de pacotes, ignorando cenários mais abrangentes (com milhares de sensores) e as restrições de recursos computacionais dos dispositivos. Além disso, Luzuriaga et al. (2017) e Castellano et al. (2018) utilizaram o IBR-DTN, que armazena os dados em RAM, ou seja, de maneira não persistente.

Moura e Hutchison (2020) mencionaram que os dispositivos de *edge computing* podem "*sintetizar dados brutos*" para reduzir o volume de dados na IoT. Alguns trabalhos avaliaram técnicas de redução de dados em sistemas IoT [Spiegel et al., 2018], [Routray et al., 2020], [Gia et al., 2019], [Chandak et al., 2020], [Blalock et al., 2018]. Porém, essas investigações não avaliaram um cenário de desconexão entre os estágios de IoT, nem consideraram a compressão dos dados cifrados na bruma computacional.

4. ReMITS: *Resilient Mist-based IoT Solution*

Para aperfeiçoar a comunicação entre a bruma e a névoa computacional, essa pesquisa propõe uma solução intitulada ReMITS (*Resilient Mist-based IoT Solution*). O ReMITS implementa os mecanismos de resiliência e redução de dados do TW-IoT [Junior e Kamienski, 2021]. Além disso, ele foca na manutenção da confidencialidade dos dados e nas limitações dos recursos computacionais da bruma. A solução ReMITS foi baseada numa solução previamente apresentada, intitulada de FIRST (*Fog-based IoT Resilient System*) [Ribeiro Junior e Kamienski, 2020] e permite que os dados enviados da bruma para a névoa não sejam perdidos, mesmo sob desconexão de rede ou em eventuais interrupções da névoa.

Com o ReMITS, quando há uma falha de conexão entre a bruma e a névoa, os dados persistem na bruma e quando a conexão retorna, ela envia os dados armazenados para a névoa. O ReMITS também reduz o volume de dados que ficam armazenados na bruma, através de algoritmos de compressão de dados e por conseguinte, reduz o tempo de envio dos dados (compactados na bruma) para a névoa, no momento em que é retomada a conexão.

A solução utiliza o mecanismo de compressão para reduzir o tamanho dos dados, pois essa técnica mantém os dados originais cifrados e sem perdas na carga útil dos pacotes. No ReMITS é possível utilizar quaisquer algoritmos clássicos de compressão de arquivos, como o bz2, Zstdandard, gzip, zlib ou lzma. Contudo, nesse trabalho também é proposto um algoritmo intitulado LoRa-SAX, que comprime especificamente os pacotes recebidos pelo ReMITS.

4.1. Fluxo de Dados do ReMITS

O ReMITS foi implementado em linguagem de programação Python e utiliza o *Mosquitto* MQTT[7] e a biblioteca *pqueue*[8], além das bibliotecas que provém os algoritmos utilizados na compressão dos dados. O fluxo de dados do ReMITS possui quatro módulos de software: *Verificador, Enfileirador, Compressor e Transmissor* (Figura 2). A solução também possui dois caminhos de dados que atuam durante a disponibilidade da conexão (*Conexão ON*) e durante a indisponibilidade da mesma (*Conexão OFF*), assim, o ReMITS garante a resiliência dos dados, mesmo com a indisponibilidade da comunicação entre a bruma e a névoa.

Como prova de conceito, foi considerada uma rede LoRaWAN[9] para exemplificar o fluxo dos dados até o ReMITS, nessa rede, os dados partem dos sensores e chegam na bruma através do *Packet Forwarder* e *LoRa Gateway Bridge* [Queté et al., 2020]. O ReMITS recebe os pacotes do *LoRa Gateway Bridge* via MQTT através do *Enfileirador* e esses pacotes são armazenados numa fila persistente chamada de *Fila DB*, sendo que o *Enfileirador* armazena os dados na *Fila DB* independentemente de eventuais desconexões ou interrupções da névoa.

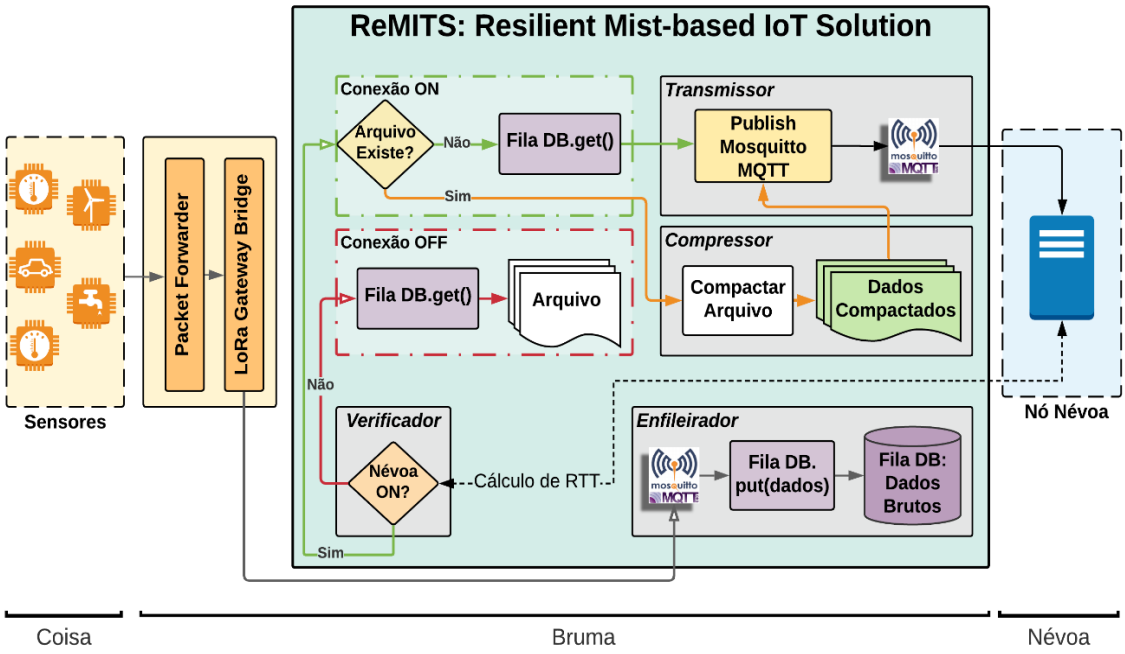

Figura 2. Fluxo da Solução ReMITS.

O *Verificador* avalia constantemente se o canal de comunicação entre a bruma e a névoa está disponível através do cálculo de RTT (*round trip time*). Assumindo que a conexão está disponível, os dados são lidos da *Fila DB* e enviados a névoa pelo *Transmissor* (via MQTT). Assumindo que a comunicação com a névoa está indisponível, os dados são lidos da *Fila DB* e continuamente armazenados em um *Arquivo*, até que a conexão retorne. Isso porque o *Enfileirador* está sempre em execução e por isso, novos dados sempre chegam à *Fila DB*. Logo, quando a conexão com a névoa retorna, o arquivo é compactado no *Compressor* e os dados compactados são enviados à névoa (via MQTT).

[7] https://mosquitto.org/
[8] https://pypi.org/project/pqueue/
[9] https://lora-alliance.org/

4.2. Algoritmo LoRa-SAX

Esse artigo também propõe o algoritmo LoRa-SAX[10] como um dos mecanismos de compressão de dados do ReMITS, sendo o LoRa-SAX baseado na técnica de *Symbolic Aggregate ApproXimation* (SAX) [Mahalakshmi e Kannan, 2016]. A técnica SAX reduz o tamanho dos dados ao agregar dados com valores semelhantes em símbolos, ela pode por exemplo, substituir dados numéricos de temperaturas de uma mesma faixa de valores em um único símbolo. Contudo, apesar de se basear no conceito da técnica SAX, o LoRa-SAX não agrega valores distintos em um único símbolo, pois no LoRa-SAX um único símbolo representa apenas uma única cadeia de caracteres.

O algoritmo LoRa-SAX funciona como um dicionário de caracteres. Ele extrai a carga útil dos pacotes LoRa, armazena a carga útil de cada pacote em um *array* e compara se existem caracteres idênticos na carga útil de cada pacote LoRa, para gerar um símbolo que represente aquele conjunto de caracteres. Ao final da execução do LoRa-SAX, um dicionário com os símbolos é gerado e inserido nas primeiras linhas de um arquivo de saída, já as demais linhas desse arquivo contêm a carga útil dos pacotes concatenada aos símbolos do dicionário.

Na descompactação (feita pela névoa), o LoRa-SAX lê o dicionário e substitui todos os símbolos pelos caracteres correspondentes, depois, o pacote LoRa é gerado de maneira padronizada na névoa e em seguida, é concatenado a carga útil. Como o pacote é remontado de maneira padronizada, existe perda de dados com relação às informações do pacote, mas não das informações contidas na carga útil do pacote (que se mantém originais). As principais informações perdidas são o número do *gateway* e o *timestamp* do pacote LoRa, mas isso é facilmente obtido caso a carga útil contenha o *timestamp* armazenado e caso (na compactação) a bruma sempre nomeie os arquivos compactados com número do *gateway* (que é conhecido pela própria bruma).

5. Metodologia

Essa seção apresenta a metodologia utilizada nas avaliações.

5.1. Ambiente de Avaliação

O ambiente de avaliação utilizado foi baseado em um sistema de fazenda inteligente [Kamienski et al., 2019] que contém a bruma e a névoa computacional (Figura 3). Sendo esse ambiente composto por quatro estágios de IoT: (i) Coisa: onde o simulador SenSE [Zyrianoff et al., 2017] gera a carga de trabalho dos sensores através da criação de pacotes LoRa, (ii) Bruma: onde o SenSE abstrai o *Packet Forwarder* e o *LoRa Gateway Bridge* [Queté et al., 2020] e envia os dados à solução ReMITS, (iii) Névoa: representando um nó de névoa que recebe dados da bruma e (iv) Nuvem: que analisa os dados enviados pela névoa.

Para o ambiente de avaliação foram utilizadas três máquinas físicas, conectadas em rede com um RTT médio de 5,67ms ± 3,3ms. O simulador SenSE foi executado em um sistema operacional Linux Ubuntu 18.04, com um processador Intel i5-8265U de 1,60 GHz e 8 GB de RAM. A solução ReMITS foi executada em um dispositivo Raspberry Pi 4B, com 4 GB de RAM e um processador ARM Cortex-A72. Já a nuvem foi executada em um sistema operacional Linux Ubuntu 18.04, com quatro núcleos de CPU de 2,4 GHz e 8 GB de RAM.

---

[10] O código do algoritmo LoRa-SAX está disponível em: https://github.com/FranklinMRJ/LoRa-SAX

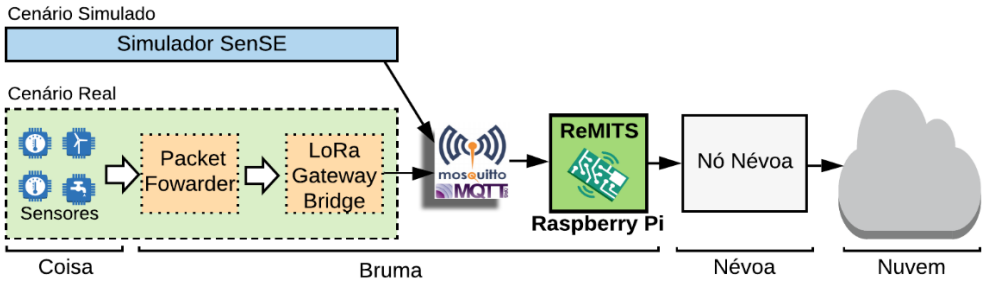

Figura 3. Ambiente Experimental.

5.2. Métricas

Na avaliação da comunicação entre a bruma e a névoa computacional, foram avaliadas as seguintes métricas:

- Tamanho dos dados: Essa métrica é aferida pelo tamanho de espaço utilizado (em disco ou SSD) pelos dados na bruma.
- Tempo de chegada: Essa métrica é calculada pela subtração do *timestamp* do último pacote a chegar na névoa, pelo *timestamp* do primeiro pacote a chegar na névoa. Ela representa o tempo que os pacotes armazenados pelo ReMITS na bruma levam para chegar na névoa após o retorno da conexão de rede.
- Tempo de compressão: Essa métrica representa o tempo que um algoritmo de compressão leva para compactar os dados na bruma. Ela é calculada pela subtração do *timestamp* aferido imediatamente antes da execução do algoritmo de compressão, pelo *timestamp* aferido imediatamente após a compactação dos dados.
- Tempo de descompressão: Essa métrica representa o tempo que um algoritmo leva para descompactar os dados na névoa. Ela é calculada pela subtração do *timestamp* aferido imediatamente antes da execução da descompressão, pelo *timestamp* aferido imediatamente após a descompressão dos dados.
- Uso de CPU: O uso de CPU na bruma foi medido a cada segundo para os processos do ReMITS, usando o comando Linux *ps -aux*.
- Uso de RAM: O uso de RAM na bruma foi medido a cada segundo para os processos do ReMITS, usando o comando Linux *ps -aux*.
- Taxa de perda de pacotes: Calculada pela comparação entre o número de pacotes que chegam na bruma e o número de pacotes que chegam à névoa.

5.3. Carga de Trabalho

Para simular a coleta de dados reais, foi utilizada uma base de dados que contém os dados climáticos de temperatura e umidade do ar da região de Sória, na Espanha [Aguilar, 2014]. A partir dos dados originais da Espanha foi gerada uma nova base de dados, onde para cada registro de um minuto foram inseridos 499 registros artificialmente, gerando uma nova base de dados com 5.000 registros (representando 5.000 sensores) a cada 10 minutos, sendo essa a carga de trabalho adotada nos experimentos.

Para criar a nova base de dados foi executado um algoritmo que gera os dados a partir de uma distribuição gaussiana entre dois registros consecutivos da base de dados original. Ao final da execução, os dados foram armazenados num arquivo CSV. Durante

a execução do experimento, os dados do arquivo são lidos pelo simulador SenSE e então, o SenSE gera pacotes com tamanho de aproximadamente 480 bytes, contendo o *timestamp* de criação do pacote e os registros de temperatura e umidade do ar.

5.4. Fluxo de Dados Avaliado e Algoritmos de Compressão Utilizados

O ReMITS foi avaliado com 7 configurações distintas: seis configurações com algoritmos de compressão e uma configuração sem nenhum tipo de compressão. Na avaliação de todas as sete configurações, a rede entre a bruma e névoa ficou inicialmente conectada por 10 segundos e após esse tempo, ela ficou desconectada em três situações, durante (a): um minuto, (b): cinco minutos e (c): 30 minutos. Após a desconexão, a comunicação foi retomada na situação: (a) 20 segundos, em (b) um minuto e em (c) cinco minutos.

Na avaliação das configurações do ReMITS foram considerados dois fluxos de dados partindo do simulador SenSE até a névoa, cujo primeiro utilizou algoritmos de compressão de dados (Figura 4) e o segundo não utilizou nenhum tipo de compressão de dados após a retomada da conexão (Figura 5). As 7 configurações utilizadas foram:

- *LoRa-SAX+BZIP2*: utiliza o algoritmo de compressão LoRa-SAX (proposto neste artigo) e em seguida o algoritmo de compressão bzip2;
- *BZIP2*: utiliza o algoritmo de compressão bzip2;
- *LZMA*: utiliza o algoritmo de compressão LZMA (*Lempel–Ziv–Markov chain algorithm*);
- *ZLIB*: utiliza o algoritmo de compressão zlib;
- *GZIP*: utiliza o algoritmo de compressão gzip;
- *ZSTD*: utiliza o algoritmo de compressão Zstandard;
- *Sem compactação*: não utiliza nenhum tipo de compressão de dados (Figura 5).

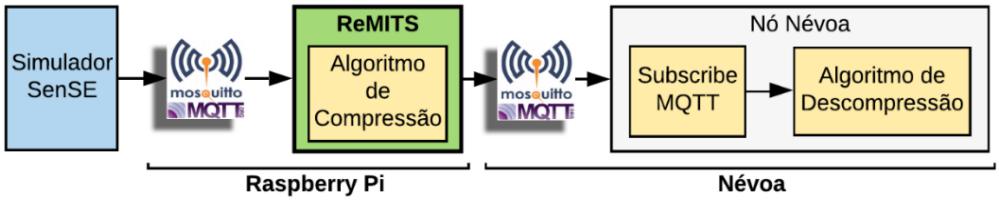

Figura 4. Fluxo com Algoritmos de Compressão de Dados Durante a Desconexão

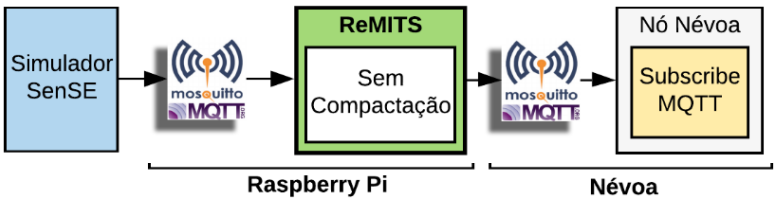

Figura 5. Fluxo sem Nenhuma Compactação de Dados Durante a Desconexão

A avaliação de cada configuração, em cada uma das situações de desconexão foi replicada 30 vezes. A metodologia não considerou uma configuração com a compressão duplicada dos dados para os algoritmos gzip, bzip2, lzma, zlib e Zstandard, já que testes preliminares mostraram que o tamanho dos dados permaneceu similar. Enquanto que o algoritmo LoRa-SAX combinado a outro algoritmo de compressão de dados (bzip2), apresentou variação nos resultados.

6. Resultados

Após executados os experimentos foram obtidos os resultados de cada configuração. Como resultado foi utilizada a média das médias (dos 30 experimentos replicados de cada configuração), além disso, foram obtidos os seus respectivos intervalos de confiança (margem de erro), para um nível de confiança de 95%.

Foi observado que as configurações que utilizaram a compactação de dados reduziram o tempo de chegada dos dados na névoa em todos os cenários de desconexão, com uma redução de 96,8% até 97,9% para um minuto de desconexão, de 97,4% até 98,3% com cinco minutos de desconexão e de 96,9% até 98,5% para 30 minutos de desconexão, em comparação a configuração sem compactação (Figura 6). Com relação ao tempo de chegada, a configuração *LoRa-SAX+BZIP2* apresentou o menor tempo, sendo 22,4% menor que o tempo da configuração *BZIP2* em 30 minutos de desconexão.

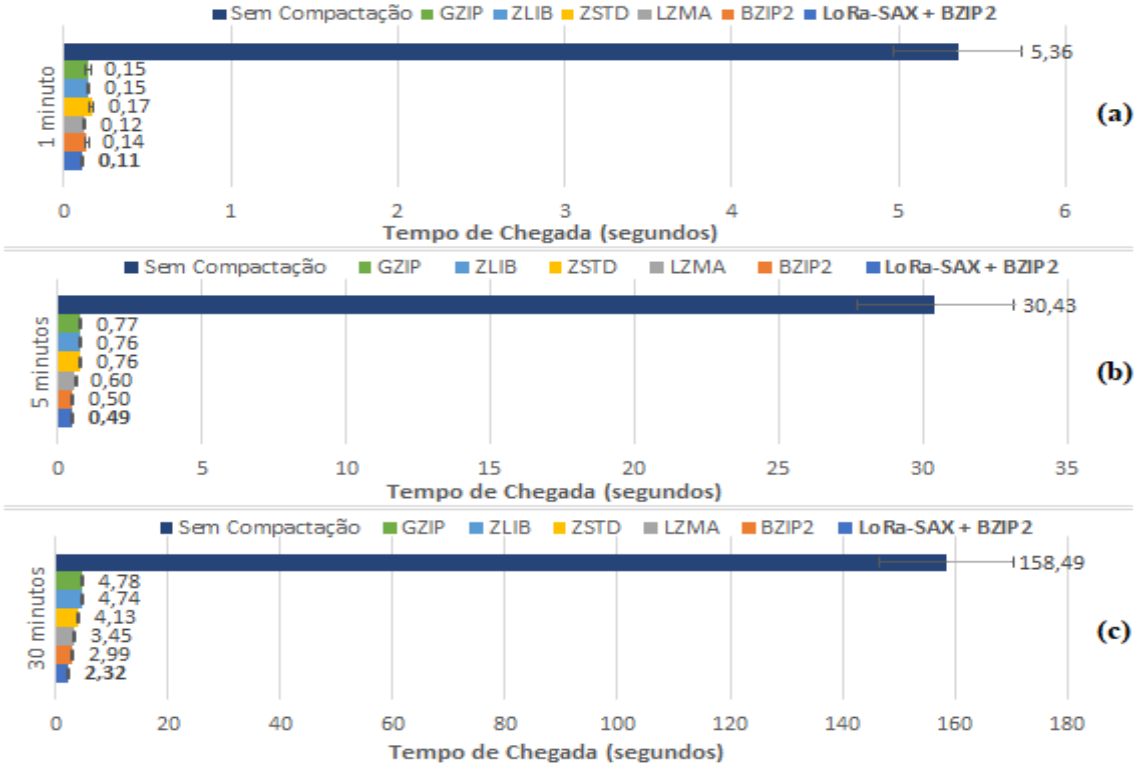

Figura 6. Tempo de Chegada dos Dados a Névoa (em segundos) após (a) um minuto, (b) cinco minutos e (c) 30 minutos de desconexão (escalas diferentes).

Com relação ao tamanho dos dados, foi percebido que a configuração sem compactação demandou mais espaço de armazenamento que as demais configurações (Figura 7). Logo, foi observada uma redução no tamanho dos dados de 89,4% até 91,3% para um minuto de desconexão, de 89,7% até 92,8% para cinco minutos de desconexão e de 90,0% até 93,6% para 30 minutos de desconexão, em comparação a configuração sem compactação. Foi observado que a configuração *LoRa-SAX+BZIP2* reportou o menor tamanho dos dados em todas as situações avaliadas e que ela apresentou o maior impacto na redução do tamanho dos dados quando o tempo de desconexão aumentou, com ganhos de aproximadamente 3%, 7,4% e 13,8% em relação a configuração *BZIP2*, para as situações de 1, 5 e 30 minutos de desconexão, respectivamente (Figura 7).

O tamanho dos dados na configuração *LoRa-SAX+BZIP2* foi menor que as demais, porque o algoritmo LoRa-SAX reduz o tamanho dos dados originais antes de utilizar o algoritmo bzip2 para compactá-los, logo, o bzip2 compacta um arquivo menor

(previamente compactado pelo LoRa-SAX), sendo que as demais configurações comprimem os dados brutos (com tamanho maior).

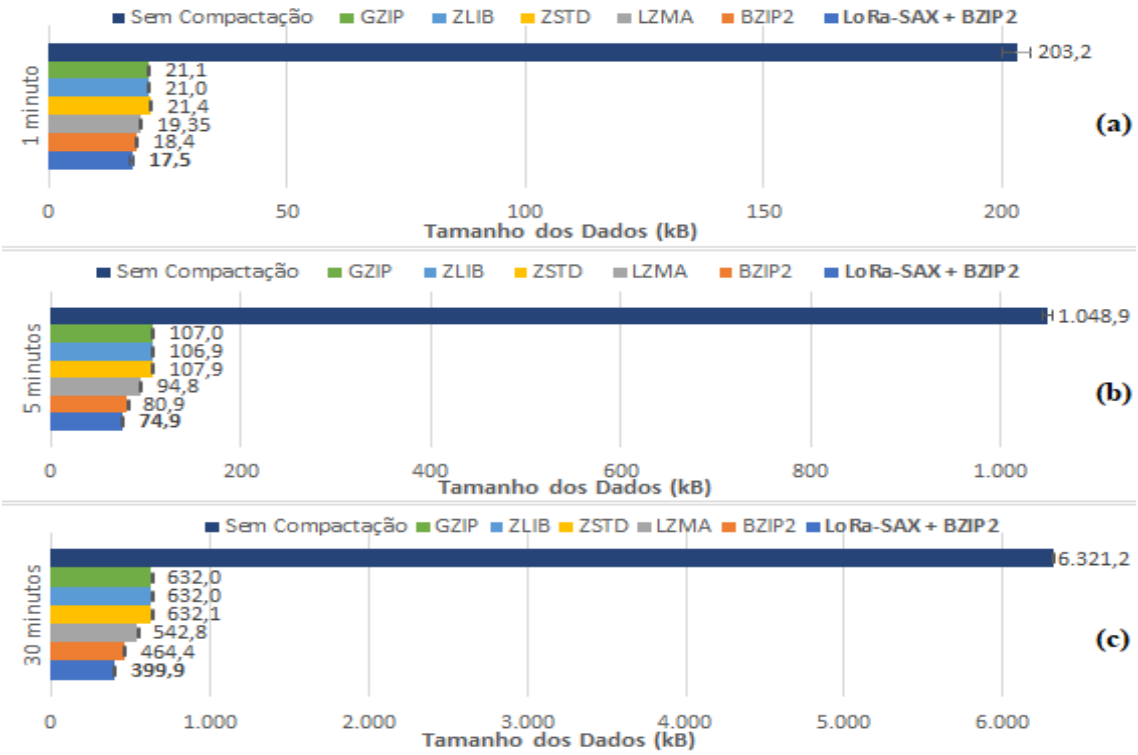

Figura 7. Tamanho dos Dados (em kilobytes) para (a) um minuto, (b) cinco minutos e (c) 30 minutos de desconexão (escalas diferentes).

Em todas as situações de desconexão, a configuração *ZSTD* levou menos tempo para compactar os dados na bruma (Figura 8). Também foi observado que todas as configurações apresentaram um aumento no tempo de compressão quando se aumentou o tempo de desconexão. Já a configuração *LoRa-SAX+BZIP2* apresentou o maior tempo de compressão (6,73 ± 0,13 segundos), com 30 minutos de desconexão (Figura 8c), ainda assim, um tempo significativamente inferior ao tempo de chegada de 158,49 ± 11,84 segundos, referente à configuração sem compactação (Figura 6c).

Foi observado que na desconexão de um minuto, a configuração *LoRa-SAX + BZIP2* apresentou tempo de compressão menor que *BZIP2* e *LZMA* (Figura 8a), mas com 5 minutos de desconexão, *LoRa-SAX+BZIP2* ficou tecnicamente empatada com *LZMA* (Figura 8b). O aumento no tempo de compressão da configuração *LoRa-SAX+BZIP2* ocorre porque o dicionário do LoRa-SAX é gerado durante a execução do algoritmo, de acordo com os dados de entrada, logo, quanto maior o volume de dados, mais símbolos o algoritmo LoRa-SAX precisa gerar, comparando mais pacotes, caractere por caractere.

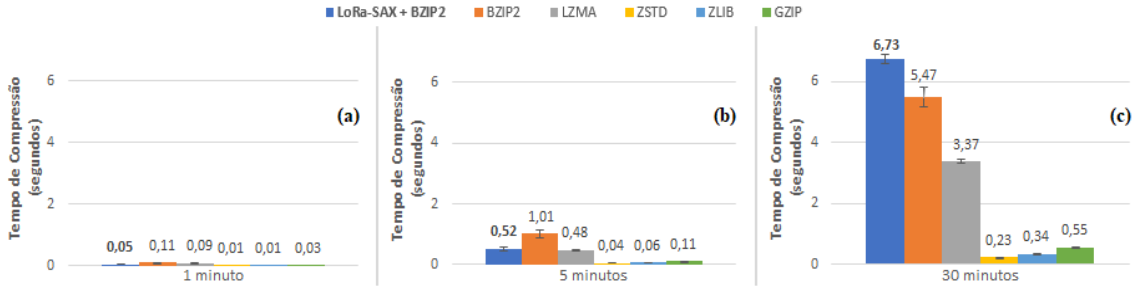

Figura 8. Tempo de Compressão (em segundos) para (a) um minuto, (b) cinco minutos e (c) 30 minutos de desconexão (mesma escala).

Foi observado que a configuração *ZSTD* apresentou o menor tempo de descompressão em todas as situações, e que todas as configurações apresentaram um tempo de descompressão inferior ao tempo de compressão (Figura 9). Um dos motivos é a descompressão executar na névoa, que possui capacidade computacional maior que a bruma. Na configuração *LoRa-SAX+BZIP2*, o tempo de descompressão é menor porque o algoritmo LoRa-SAX apenas substitui os símbolos do arquivo compactado pelos símbolos do dicionário (obtidos durante a compressão). Logo, ele realiza menos comparações durante o processo de descompressão dos dados.

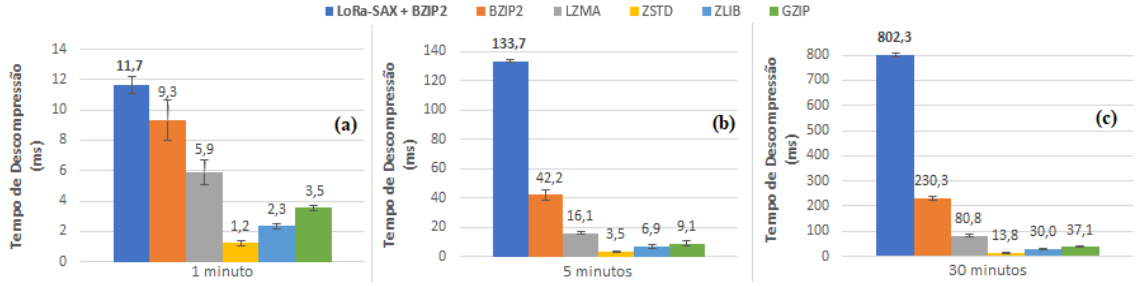

Figura 9. Tempo de Descompressão (em milissegundos) para (a) um minuto, (b) cinco minutos e (c) 30 minutos de desconexão (escalas diferentes).

O uso de CPU da configuração sem compactação foi, em média, menor que o das configurações *LoRa-SAX+BZIP2* e *LZMA*, porém todas as demais configurações ficaram tecnicamente empatadas, considerando o intervalo de confiança de 95% (Figura 10). Em todas as situações de desconexão, o uso de RAM foi de aproximadamente 0,4% ± 6,06e-17 para todas as configurações. Além disso, não foi observada perda de pacotes em nenhum dos experimentos. Logo, a taxa de perda de pacotes foi de 0%.

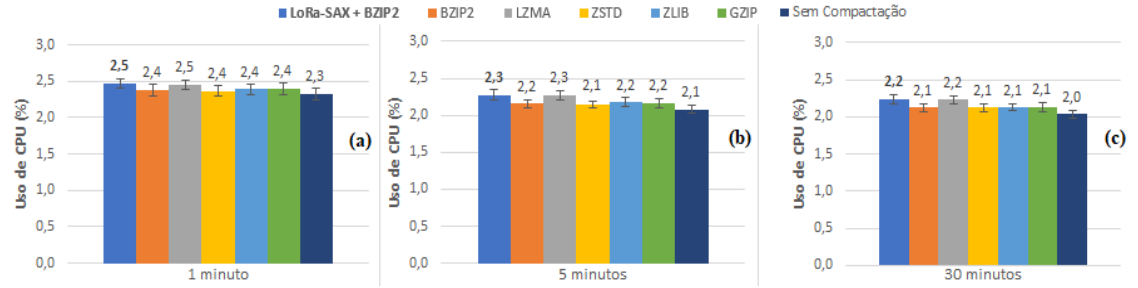

Figura 10. Uso de CPU para (a) um minuto, (b) cinco minutos e (c) 30 minutos de desconexão.

7. Discussão

Após analisados os resultados, foi possível perceber que o ReMITS reduziu o tempo de chegada dos dados na névoa com um custo adicional para a bruma de 0,4% de RAM, e entre 2% e 2,5% de CPU. Também foi constatado que uso do ReMITS armazena os pacotes cifrados de maneira persistente na bruma e envia todos os pacotes à nevoa, já que não houve perda de pacotes entre a bruma e a névoa após situações de desconexão de rede.

Foi observado que uso da compressão no ReMITS impacta o fluxo de dados do sistema IoT após a retomada de uma desconexão, uma vez que o sistema levou no mínimo 96,8% menos tempo para que todos os dados chegassem à névoa, reduzindo o atraso no envio de futuros pacotes. Essa constatação torna-se evidente, pois com 30 minutos de desconexão, sem o uso da compactação dos dados, a solução levou, em média, quase 2 minutos e 37 segundos para que os dados chegassem à névoa, contra 4,78 segundos da

configuração *GZIP*, por exemplo. Portanto, há evidências de que, sem a compressão dos dados, em uma desconexão de várias horas, os pacotes armazenados na bruma levariam muito mais tempo para ser enviados a névoa, causando um atraso significativo no envio de futuros pacotes.

Foi constatado que não existe uma configuração ótima para todas as métricas. Apesar da configuração *LoRa-SAX+BZIP2* reportar o menor tamanho dos dados e o menor tempo de chegada dos dados a névoa, ela obteve o maior tempo de compressão na bruma e o maior tempo de descompressão na névoa, com 30 minutos de desconexão. Enquanto que a configuração *ZSTD* apresentou menor tempo de compressão na bruma e de descompressão na névoa, porém a configuração *ZSTD* apresentou o maior valor para o tamanho dos dados.

Nesse trabalho foi observado um *trade-off* entre a métrica de tamanho dos dados contra o somatório dos tempos de compressão, de descompressão e no tempo de chegada dos dados a névoa. Em uma desconexão de 30 minutos, o *LoRa-SAX+BZIP2* apresentou uma redução de 36,7% no tamanho dos dados e 43,8% menos tempo de chegada dos dados à névoa em relação a configuração *ZSTD*. Porém, com desconexão de 30 minutos, a configuração *ZSTD* levou 96,6% menos tempo para comprimir os dados e 98,2% menos tempo para descompactar os dados, sendo que na configuração *ZSTD* o somatório dos tempos (de compressão, descompressão e de chegada dos dados) foi menor que o mesmo somatório dos tempos para a configuração *LoRa-SAX+BZIP2*. Portanto, a escolha pelo uso de cada configuração depende dos requisitos do sistema IoT.

8. Conclusão

Este trabalho apresentou um estudo sobre o impacto da resiliência e da redução no tamanho dos dados, na comunicação entre a bruma e a névoa computacional na IoT. Como conceito de prova, foi proposta e avaliada uma solução intitulada ReMITS, que executa na bruma computacional. O ReMITS armazena os dados de maneira persistente, mesmo em situações de desconexão de rede ou de interrupção da névoa, além de compactar os dados quando é retomada a comunicação. O ReMITS foi avaliado em uma placa *Raspberry Pi 4B* com diferentes algoritmos de compressão de dados contra um algoritmo proposto, intitulado de LoRa-SAX.

Os resultados da avaliação mostraram que houve um ganho de pelo menos 89,4% na redução do tamanho dos dados e de pelo menos 96,8% na redução do tempo de chegada dos dados a névoa. Com o uso do LoRa-SAX combinado ao bzip2, o ReMITS reduziu ainda mais o tamanho dos dados quando comparado aos demais algoritmos de compressão, com reduções de 91,3% em até 93,6%. Também foi observado que o ReMITS somente utilizou 0,4% de RAM e entre 2% a 2,5% de CPU da bruma.

O algoritmo LoRa-SAX combinado ao bzip2 também apresentou o menor tempo de chegada, porém como o algoritmo Zstandard apresentou o menor tempo de compressão na bruma e descompressão na névoa, o Zstandard pode ser mais vantajoso ao considerarmos o somatório de todas as métricas de tempo observadas. Como trabalho futuro, espera-se aperfeiçoar a proposta para avaliar mais requisitos de *trustworthiness* em nível de dados, em um sistema IoT baseado em bruma e névoa computacional.

Referências

Aguilar F. (2014), Press, Temperature and Humidity August 2013, DataONE Dash, Dataset, https://doi.org/10.15146/R3730R.

Al-Khafajiy M., Baker T., Waraich A., Al-Jumeily D. and Hussain A., (2018) "IoT-Fog Optimal Workload via Fog Offloading," 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018.

Armbrust M., Stoica I., Zaharia M., Fox A., Griffith R., Joseph A. D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A. (2010) "A View of Cloud Computing". Commun. Acm, Vol. 53, No. 4, Pp. 50–58, 2010.

Asif-Ur-Rahman, Afsana F., Mahmud M., Shamim Kaiser M., Ahmed M. R., Kaiwartya O. and James-Taylor A. (2019) "Toward a Heterogeneous Mist, Fog, and Cloud-Based Framework for the Internet of Healthcare Things", in IEEE Internet of Things Journal.

Atlam H. F., Walters R. J., Wills G.B. (2018) "Fog Computing and the Internet of Things: A Review". Big Data Cogn. Comput.

Atzori L., Iera A., and Morabito G. (2010) "The internet of things: A survey". Computer Networks, 54(15):2787-2805.

Azar J., Makhoul A., Barhamgi M., Couturier R. (2019) "An energy efficient IoT data compression approach for edge machine learning", Future Generation Computer Systems, Volume 96, 2019, Pages 168-175, ISSN 0167-739X.

Blalock D., Madden S., and Guttag J. (2018) "Sprintz: Time Series Compression for the Internet of Things". Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2, 3, Article 93 (September 2018), 23 pages.

Castellano G., Risso F. and Loti R. (2018) "Fog Computing Over Challenged Networks: A Real Case Evaluation", 2018 IEEE (CloudNet), Tokyo, 2018.

Chandak S., Tatwawadi K., Wen C., Wang L., Ojea J. A., Weissman T. (2020) "LFZip: Lossy Compression of Multivariate Floating-Point Time Series Data via Improved Prediction", 2020 Data Compression Conference (DCC), Snowbird, UT, USA, 2020.

Cho J., Xu S., Hurley P. M., Mackay M., Benjamin T., and Beaumont M. (2019) "STRAM: Measuring the Trustworthiness of Computer-Based Systems". ACM Comput. Surv. 51, 6, Article 128, 47 pages.

Fall K. (2003). A delay-tolerant network architecture for challenged internets. (SIGCOMM '03). ACM, New York, NY, USA.

Gia T. N., Jiang M., Rahmani A., Westerlund T., Liljeberg P., and Tenhunen H. (2015) "Fog Computing in Healthcare Internet of Things: A Case Study on ECG Feature Extraction", 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, Liverpool.

Gia T. N., Qingqing L., Queralta J. P., Tenhunen H., Zou Z., Westerlund T. (2019) "Lossless Compression Techniques in Edge Computing for Mission-Critical Applications in the IoT", 2019 Twelfth International Conference on Mobile Computing and Ubiquitous Network (ICMU), Kathmandu, Nepal, 2019, pp. 1-2.

Harchol Y., Mushtaq A., McCauley J., Panda A., Shenker S., (2018) "Cessna: Resilient edge computing", 2018 Workshop on Mobile Edge Communications, ACM, pp. 1–6.

Jeong T., Chung J., Hong J.W-K, Ha S., (2017) "Towards a distributed computing framework for fog", in: Fog World Congress (FWC), 2017 IEEE, IEEE, pp. 1–6.

Jonathan A., Uluyol M., Chandra A., Weissman J. (2017) "Ensuring reliability in geo-distributed edge cloud", in: Resilience Week (RWS), IEEE, 2017, pp. 127–132.

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, 2021, doi: 10.1109/ACCESS.2021.3066457.

Kamienski C., Soininen J.-P., Taumberger M., Dantas R., Toscano A., Salmon Cinotti T., Filev M. R., Torre Neto A. (2019) "Smart Water Management Platform: IoT-Based Precision Irrigation for Agriculture". Sensors, 19, 276.

Kulatunga C., Shalloo L., Donnelly W., Robson E. and Ivanov S. (2017) “Opportunistic Wireless Networking for Smart Dairy Farming”, in IT Professional, vol. 19, no. 2.

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, 2019.

Luzuriaga J. E., Zennaro M., Cano J. C., Calafate C. and Manzoni P., (2017) "A disruption tolerant architecture based on MQTT for IoT applications," 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV.

Mahalakshmi R. and Kannan S. (2016) “Semantic Filtering of IoT Data using Symbolic Aggregate Approximation (SAX)”, Journal of Computer Science and Applications, vol. 8, no. 1, pp. 31-39, 2016.

Moura J., Hutchison D. (2020) “Fog computing systems: State of the art, Research issues and future trends, with a focus on resilience”, Journal of Network and Computer Applications, Volume 169, 2020, 102784, ISSN 1084-8045.

Negash B., Gia T.N., Anzanpour A., Azimi I., Jiang M., Westerlund T., Rahmani A.M., Liljeberg P., Tenhunen H. (2018) Leveraging Fog Computing for Healthcare IoT. In: Rahmani A., Liljeberg P., Preden JS., Jantsch A. (eds) Fog Computing in the Internet of Things. Springer, Cham.

Preden J. S., Tammemäe K., Jantsch A., Leier M., Riid A. and Calis E. (2015) “The Benefits of Self-Awareness and Attention in Fog and Mist Computing”, in *Computer*.

Queté B., Heideker A., Zyrianoff, I., Ottolini D., Kleinschmidt J.H., Soininen J-P., Kamienski C. (2020) “Understanding the tradeoffs of LoRaWAN for IoT-based Smart Irrigation”, 2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor), Trento, Italy, 2020, pp. 73-77.

Routray S. K., Javali A., Sahoo A., Semunigus W., Pappa M. (2020) “Lossless Compression Techniques for Low Bandwidth IoTs”, 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2020, pp. 177-181.

Ribeiro Junior F., Kamienski C. (2020) “Resiliência de Dados entre a Névoa e a Nuvem na Internet das Coisas”, in Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, Rio de Janeiro, 2020, pp. 85-98.

Ribeiro F. M., Prati R., Bianchi R., Kamienski C. (2020) “A Nearest Neighbors based Data Filter for Fog Computing in IoT Smart Agriculture”, 2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor), Trento, 2020.

Shi Y., Ding G., Wang H., Roman H. E., Lu S. (2015) “The fog computing service for healthcare”. 2nd International Symposium on Future Information and Communication Technologies for Ubiquitous HealthCare (Ubi-HealthTech), Beijing, 2015, pp. 1-5.

Spiegel J., Wira P., and Hermann G. (2018) “A Comparative Experimental Study of Lossless Compression Algorithms for Enhancing Energy Efficiency in Smart Meters”, 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), Porto.

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

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), 2017.

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, Dec. 2019.