Resiliência de Dados entre a Névoa e a Nuvem 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

Resumo. *A computação em névoa permite processar e analisar os dados recebidos de sensores com maior agilidade na borda de uma rede IoT. Contudo, a nuvem ainda é necessária para centralizar e analisar dados históricos, já que a névoa possui recursos limitados. Nesse cenário, desconexões de rede entre a névoa e a nuvem podem gerar perdas de dados, comprometendo análises futuras. Esse artigo propõe e avalia técnicas para manter a resiliência dos dados entre névoa e nuvem em situações de desconexão. Foram avaliados o Apache Kafka, o Apache NiFi e uma solução proposta, intitulada FIRST, numa carga simulada de 1.000, 3.000 e 5.000 sensores. Foi observado que o FIRST e o Kafka apresentaram a menor latência, enquanto que FIRST manteve menor uso de CPU e RAM.*

Abstract: *Fog computing allows the processing and analysis of data with agility in an IoT network edge. However, the cloud is still needed to centralize data and allow historic data analysis that is not possible with limited fog resources. In this scenario, network disconnections between fog and cloud may generate data losses that endanger future analyses. This paper proposes and evaluates techniques for keeping data resilience between fog and cloud considering disconnections. We evaluate Apache Kafka, Apache NiFi and our proposed solution, called FIRST, with a simulated workload of 1,000, 3,000 and 5,000 sensors. We observed that FIRST and Kafka presented the lowest data transfer delay, whereas FIRST used less CPU and RAM resources.*

1. Introdução

A Internet das Coisas (*Internet of Things* - IoT) caracteriza-se por uma rede de dispositivos físicos (sensores e atuadores) que podem transmitir e receber dados em determinados contextos de aplicação, de modo a possibilitar a redução de custos em ambientes inteligentes [Atzori et al., 2010]. Em sistemas IoT, através da névoa computacional, é possível armazenar, analisar e filtrar os dados localmente na borda, sem criar dependências do sistema com a disponibilidade constante da conexão com a nuvem [Aazam et al., 2018]. Além disso, em determinados contextos, a névoa possibilita uma ação mais veloz do sistema através dos atuadores [Mouradian et al., 2018].

Contudo, existem desafios de pesquisa quanto à implantação de sua arquitetura, especialmente no caminho dos dados do sensor até a nuvem [Ray, 2018] e [Atlam et al., 2018]. Um dos desafios apontados por [Yousefpour et al., 2019], consiste no

estabelecimento de mecanismos para garantir a resiliência na transmissão de dados, quando há desconexão ou falhas na rede da névoa. No entanto, para lidar com esses mecanismos de comunicação, as soluções em névoa devem lidar com o gerenciamento de dados advindos de milhares de sensores de forma robusta, porém essas soluções devem ponderar restrições relacionadas a capacidade de memória e de processamento [Ray, 2018] e [Atlam et al., 2018].

Em ambientes de cidades e agricultura inteligente, como por exemplo, nos pilotos do projeto SWAMP [Kamienski, et al. 2019] e [Zyrianoff, et al. 2019], que utilizam, a tecnologia LoRaWAN[1], foi identificado que, em uma eventual desconexão da névoa com a nuvem, assim que retomada a conexão, os dados não enviados à nuvem (durante a desconexão), não são imediatamente reenviados, causando uma lacuna na continuidade do fluxo de dados entre a névoa e a nuvem. Logo, em situações em que há perda de conexão de rede entre a névoa e a nuvem, é necessário que os dados sejam armazenados na névoa, para que mais tarde, possam ser transmitidos à nuvem, sem que haja perda dos dados (anteriormente recebidos pela névoa), durante o período de desconexão.

Por isso, esse artigo propõe uma solução intitulada FIRST (*Fog-based IoT Resilient System*), para manter a continuidade do fluxo de dados na comunicação entre a névoa e a nuvem, de modo a atender a resiliência do sistema IoT, mencionada por [Cho et al., 2019], como um dos requisitos de *trustworthiness*[2] em sistemas computacionais. Com isso, a proposta visa manter a persistência do fluxo de dados entre a névoa e a nuvem, respeitando limitações de hardware da névoa.

No âmbito de validar a proposta apresentada, foi realizada uma avaliação de desempenho contra outras tecnologias de comunicação confiável, tais como a ferramenta Apache Kafka e a ferramenta Apache NiFi (com as configurações: MQTT e Apache Kafka). Desse modo, foram avaliadas configurações para o fluxo de dados entre a névoa e a nuvem, quanto ao uso de processamento, memória e latência, em cenários onde a conexão de rede está disponível. Já em um cenário de desconexão (com a retomada do fluxo), foi aferido o tempo de retomada da conexão e o tempo de envio de todos os pacotes à nuvem (desde a retomada da conexão e desde a desconexão).

Após as avaliações, os resultados obtidos apontaram que as configurações que utilizaram o Apache NiFi apresentaram maior uso de CPU e memória de RAM, enquanto que a configuração que utilizou o FIRST ou que somente utilizou as ferramentas do Apache Kafka apresentaram menores valores de latência e do tempo de retomada da conexão. Ademais, foi observado que, com o FIRST foi possível atender a resiliência, com menor uso de recursos computacionais da névoa (CPU e RAM).

O restante do trabalho está organizado da seguinte forma: a Seção 2 apresenta a fundamentação, a Seção 3 discute os trabalhos relacionados, a Seção 4 apresenta a solução proposta (FIRST), a Seção 5 discorre sobre a metodologia utilizada, a Seção 6 evidencia os resultados, a Seção 7 discute os resultados e, finalmente, a Seção 8 relata as conclusões e os trabalhos futuros, com base nas evidências encontradas.

---

[1] https://lora-alliance.org/

[2] O termo *trustworthiness* foi mantido em inglês por não haver uma palavra similar em português, amplamente aceita pela comunidade acadêmica.

2. Fundamentação

Essa seção apresenta a fundamentação.

2.1. Computação em Névoa e em Bruma

Com o uso da névoa computacional, é possível ao sistema IoT, realizar uma análise mais veloz dos dados, devido ao ganho de recursos computacionais na borda. Para implantar a névoa, o sistema IoT deve possuir uma arquitetura, que geralmente é dividida em três camadas [Aazam et al., 2018], [Chiang; Zhang, 2016], [Mukherjee et al, 2018]: (i) a primeira camada é composta por sensores, atuadores e/ou dispositivos que permitem conectar os usuários a névoa, (ii) a segunda camada é composta pela névoa, compreendida por computadores, switches, roteadores e pontos de acesso, onde pode ser realizado o processamento dos dados coletados e finalmente, (iii) a terceira camada é composta pela nuvem, que está conectada a camada da névoa através da internet.

A computação em bruma[3] (*mist computing*) caracteriza-se por estar mais próxima dos sensores e pode ser entendida, como uma subdivisão da névoa, de modo que pode realizar as mesmas tarefas da névoa, mas de forma limitada [Asif-Ur-Rahman et al., 2019], [Linaje et al., 2019]. A sua vantagem pode ser observada na escalabilidade do sistema, uma vez que, através da bruma é possível aumentar a autonomia dos dispositivos mais próximos à borda [Preden et al., 2015].

Para lidar com o desafio de atender a um grande fluxo de dados, a bruma e a névoa devem ter arquiteturas robustas, porém devem considerar as suas respectivas limitações de recursos computacionais. Cho et al. (2019) definiram *trustworthiness* como um arcabouço que lida com um conjunto de requisitos para garantir a robustez de um sistema computacional. Esse conjunto, por exemplo, compreende os requisitos de segurança, confiança, agilidade e resiliência do sistema. Contudo, o conceito de *trustworthiness* não foi concebido para lidar especificamente com requisitos em nível de dados em IoT, como pode ser percebido no trabalho de revisão apresentado por [Cho et al., 2019].

2.2. Tecnologias Confiáveis no Fluxo Névoa-Nuvem

Dentre as tecnologias utilizadas nas configurações dos fluxos avaliados, estão o Mosquitto[4] MQTT, o Apache Kafka[5] e o Apache NiFi[6]. O MQTT[7] (*Message Queuing Telemetry Transport*) é um protocolo de mensagens que utiliza o padrão de comunicação *publish*/*subscribe* (pub/sub), que executa sobre a arquitetura TCP/IP e que, é comumente utilizado em sistemas IoT. Já o Mosquitto MQTT é uma plataforma que suporta o MQTT, pela qual, é possível publicar e subscrever mensagens à um MQTT Broker.

O Apache Kafka [Kreps et al., 2011] é citado por [Ivaki et al., 2018] como um sistema de comunicação persistente e confiável. Ele consiste numa plataforma de mensageria, a qual possui 5 elementos principais: (i) o Kafka Broker, que armazena as mensagens de um ou mais tópicos, (ii) os tópicos, aos quais estão endereçadas as

---

[3] *O termo mist computing ainda é* novo e não completamente adotado pela comunidade científica. Este artigo propõe a tradução "computação em bruma" para *mist computing*.

[4] https://mosquitto.org/

[5] https://kafka.apache.org/

[6] https://nifi.apache.org/

[7] http://mqtt.org/

mensagens, (iii) o produtor, que é responsável por enviar as mensagens a um determinado tópico, (iv) o consumidor, que consome as mensagens de um tópico e (v) o Apache Zookeeper, que controla o acesso aos tópicos do Kafka Broker.

No conjunto de ferramentas do Kafka, também existe o Kafka Mirror Maker, que é responsável por replicar os dados de um ou mais tópicos de um Kafka Broker (1) a outro Kafka Broker (2). Assim, o Kafka Mirror Maker realiza um fluxo consumidor-produtor, cujas mensagens são consumidas de (1) e enviadas pelo produtor à (2). Adicionalmente foi utilizada a ferramenta Kafka MQTT Connect [Evokly, 2016], que permite escutar os dados de um MQTT Broker e publicá-los no Kafka Broker.

O Apache NiFi é uma ferramenta de suporte a elaboração de fluxo de dados. O NiFi suporta diversos componentes de ingestão de dados, assim como um pub/sub MQTT, bem como os módulos produtor e consumidor do Kafka. Com ele, é também possível transmitir mensagens entre os diferentes componentes suportados. Adicionalmente, nos trabalhos em [Dautov et al., 2018] e [Ravindra, et al., 2017] foram propostas plataformas para o fluxo de dados entre a borda e a nuvem com a utilização do Apache NiFi, como ferramenta de middleware, para lidar com o *pipeline* dos dados.

3. Trabalhos Relacionados

Nas investigações discutidas em [Asif-Ur-Rahman et al., 2019] e [Linaje et al., 2019] os autores apresentaram abordagens para o fluxo de dados através da névoa computacional em sistemas IoT. Já nos trabalhos em [Jonathan et al., 2017], [Harchol et al., 2018], [Jeong et al., 2017], [Rimal et al., 2017], [Machen et al., 2018], [Helmer et al., 2016] e [Jonathan; Chandra e Weissman, 2017], os autores apresentaram estudos sobre resiliência na comunicação em sistemas de *mist*, *fog* ou *edge computing*. Já em [Zyrianoff, et al. 2019] foi realizada uma avaliação de desempenho, para diversas configurações de fluxos de dados, em plataformas IoT baseadas em névoa computacional, que utilizam a arquitetura LoRaWAN.

No entanto, apesar das contribuições encontradas, as investigações, não apresentaram em suas pesquisas, por exemplo, mecanismos de detecção e recuperação de falhas em uma provável queda na conexão da Internet (no caminho névoa-nuvem). Sendo esse um desafio destacado em [Yousefpour et al., 2019] e [Atlam et al., 2018].

Com relação as investigações relacionadas a DTN [Fall, 2003] em sistemas de névoa computacional, foram encontrados os trabalhos em [Kovalchuk et al., 2019], [Luzuriaga et al., 2017], [Kulatunga et al., 2017], [Al-khafajiy et al., 2018] e [Castellano et al., 2018]. Contudo, apesar das contribuições, nem todos os trabalhos preocuparam-se com a persistência dos dados, além disso, os trabalhos somente trataram de métricas de atraso ou de perda de pacotes, desconsiderando o impacto das soluções, frente às limitações dos recursos computacionais da névoa, isso porque, a DTN não foi projetada para lidar com os requisitos de *trustworthiness* em sistemas computacionais.

4. FIRST: Resiliência na Comunicação Névoa-Nuvem

Para aperfeiçoar a comunicação entre a névoa computacional e a nuvem, essa pesquisa baseou-se na especificação do arcabouço de *trustworthiness* para sistemas computacionais, com foco na resiliência da comunicação dos dados e nas limitações dos recursos computacionais da névoa.

4.1. Requisitos de Resiliência

A fim de atender ao requisito de resiliência, foi desenvolvida uma solução, intitulada FIRST (*Fog-based IoT Resilient System*), projetada para lidar com a (i) confiabilidade, a (ii) recuperabilidade (*survivability*) e as (iii) limitações de recursos computacionais da névoa:

(i) Para garantir a confiabilidade, os dados devem ser entregues de modo a não perder a qualidade do contexto e de análises futuras. Por isso, a maneira adotada por essa proposta, para tornar o fluxo de dados, em um sistema IoT mais confiável, foi por adotar o modelo de comunicação assíncrona, *one-way* e persistente, através da técnica *publish*/*subscribe*, com a utilização do protocolo MQTT.

(ii) Para garantir a recuperabilidade (*survivability*) dos dados na comunicação entre a névoa e a nuvem, os dados da névoa devem trafegar sem apresentar perda de pacotes, de um dispositivo a outro. Desse modo, quando houver uma falha na conexão entre névoa e nuvem, os dados devem persistir na névoa. Assim, no momento em que for restabelecida a conexão, os dados devem ser entregues à nuvem. Por isso, foi implementado um módulo que armazena os dados em fila e verifica a conexão e a desconexão com a nuvem, constantemente.

(iii) Para utilizar o mínimo possível de recursos computacionais, FIRST foi projetado com o uso de apenas duas *threads* (implementadas em linguagem de programação), de modo que, a cada etapa da implementação, foram realizados testes preliminares para verificar o uso dos recursos computacionais.

4.2. Arquitetura da Solução FIRST

Para atender e validar os requisitos de resiliência, quanto a continuidade do fluxo dos dados, no caminho névoa-nuvem, foi desenvolvida a solução FIRST, que mantém a persistência dos dados em prováveis desconexões de rede (Figura 1). Cuja solução foi escrita em linguagem de programação Python e utiliza a biblioteca *pqueue*[8] para manter os dados em disco, que persistem mesmo em uma interrupção (desligamento) da máquina da névoa.

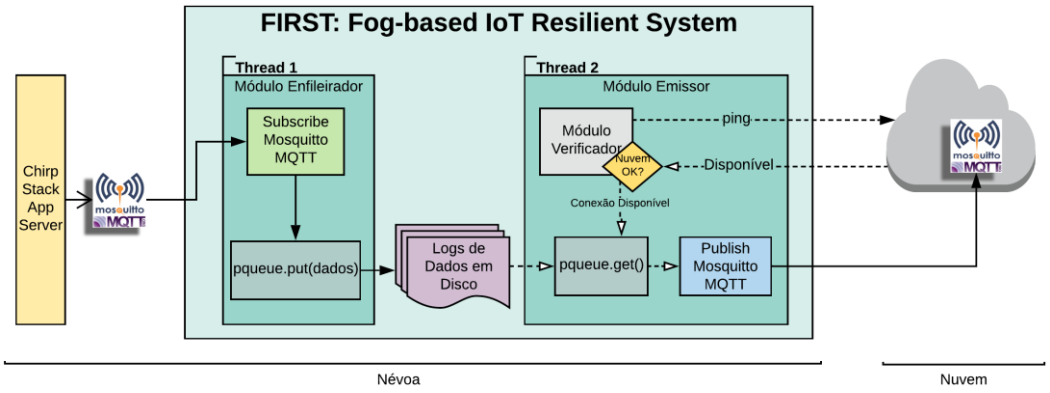


Figura 1. Arquitetura da Solução FIRST (*Fog-based IoT Resilient System*).

O FIRST executa (na névoa) e possui duas *threads*, a primeira (corresponde ao módulo enfileirador e) executa um *subscribe* (Mosquitto) MQTT, para escutar os dados enviados pelo ChirpStack App Server[9] (via MQTT), cujas mensagens, após escutadas,

[8] https://pypi.org/project/pqueue/
[9] https://www.chirpstack.io/

são armazenadas em fila através do *pqueue* (em logs de dados em disco). Já a segunda *thread* (que corresponde ao módulo emissor), verifica constantemente quando há conexão de rede com a nuvem, para que sejam enviados (à nuvem) os dados armazenados em fila (pelo *pqueue*). Tal verificação é realizada (pelo módulo verificador), através do cálculo do RTT, que verifica constantemente a disponibilidade da conexão com a nuvem.

Ainda que a conexão seja perdida, a primeira *thread* continuará armazenando os dados em fila de maneira persistente, para que mais tarde, quando houver o restabelecimento da conexão, os dados sejam enviados para a nuvem.

5. Metodologia

Essa seção apresenta a metodologia utilizada nas avaliações.

5.1. Métricas

No estudo, foram avaliadas configurações de fluxos de dados, na comunicação névoa-nuvem, com o uso das ferramentas Mosquitto MQTT, Apache Kafka, Apache NiFi e da solução proposta FIRST, em dois cenários: (i) um cenário de condições normais e (ii) em um cenário de desconexão, com a retomada da conexão. Portanto, as métricas utilizadas para o cenário (i) foram:

- Atraso (latência): Para aferir a métrica do atraso, foi registrado no payload de cada pacote LoRa, o tempo (*timestamp*) de criação do pacote. Assim, quando o pacote é consumido na nuvem, é registrado o tempo de chegada (*timestamp*), e mais tarde são subtraídos os tempos de chegada e de criação de cada pacote;
- Uso de memória RAM: Foram coletados os registros do uso de memória RAM, através do comando *ps -aux* do Linux, onde foi possível aferir, numa periodicidade de 1 (um) segundo o uso de memória RAM, de cada um dos processos executados pelo FIRST, Apache Kafka e Apache NiFi;
- Uso de CPU (processador): Foram coletados os registros do uso de CPU, através do comando *ps -aux* do Linux, onde é possível aferir, numa periodicidade de 1 (um) segundo o uso de CPU, de cada um dos processos executados pelo FIRST, Apache Kafka e Apache NiFi. Ademais, é válido ressaltar que, como a máquina utilizada na névoa do experimento possui 8 núcleos de CPU, os resultados para essa métrica foram divididos pelo número inteiro 8 (oito);
- Taxa de perda de pacotes: Para aferir a perda de pacotes foi executado um consumidor MQTT (na névoa) e um consumidor MQTT ou Kafka (na nuvem), mais tarde foram comparados todos os pacotes que partiram da névoa e que chegaram na nuvem, ao final de cada experimento.

Para o cenário (ii), além da taxa de perda de pacotes, foram utilizadas as seguintes métricas:

- Tempo de retomada da conexão (desde a desconexão): Para aferir essa métrica foi utilizado o tempo (*timestamp*) de chegada do primeiro pacote à nuvem, desde a criação do pacote, até a chegada na nuvem.
- Tempo de envio de todos os pacotes à nuvem (a partir da retomada da conexão): Para aferir essa métrica foi utilizada a subtração (do *timestamp*) do tempo de

chegada do último pacote (que chega à nuvem), pelo tempo de chegada do primeiro pacote (que chega à nuvem).

- Tempo total de envio de todos os pacotes à nuvem (desde a desconexão): Para aferir essa métrica foi utilizada a soma entre o tempo de retomada da conexão (desde a desconexão) e o tempo de envio de todos os pacotes à nuvem (a partir da retomada da conexão).

5.2. Carga de trabalho

A carga de trabalho foi gerada, através do simulador SenSE [Zyrianoff et al., 2017], que segue o modelo da distribuição de Poisson, com pacotes no formato LoRa, de tamanho fixo de aproximadamente 306 bytes.

Para os experimentos foram adotados dois cenários: (i) de disponibilidade da conexão (névoa-nuvem), onde foi avaliada uma carga de trabalho simulada de 1.000, 3.000 e 5.000 sensores e (ii) de desconexão (névoa-nuvem), onde foi avaliada uma carga simulada de 1.000 e 5.000 sensores. Em ambos os cenários, os sensores são responsáveis por enviar pacotes em um período de 10 minutos, período este, baseado no cenário real dos pilotos do projeto SWAMP [Kamienski et al., 2019].

5.3. Ambiente de Avaliação

O ambiente de avaliação seguiu a arquitetura de um sistema IoT baseado em névoa computacional, com o fluxo de dados névoa-nuvem (Figura 2), tal arquitetura é utilizada por exemplo, nos trabalhos em [Kamienski et al., 2019] e [Zyrianoff et al., 2019]. Esse ambiente possui 4 visões arquiteturais: (i) sensores que captam sinais de temperatura e umidade do solo, (ii) LoRa Gateway, que consistem em placas Raspberry Pi, representando a bruma computacional, que envia dados a, (iii) uma névoa por meio de rede local via Wi-Fi, onde estão instalados o ChirpStack Network Server e ChirpStack App Server e, finalmente (iv) a nuvem que corresponde a um servidor, onde está instalada a plataforma FIWARE[10], cujo FIWARE IoT Agent é responsável por receber os dados enviados pelo ChirpStack App Server da névoa, e por enviar os dados ao FIWARE Orion.

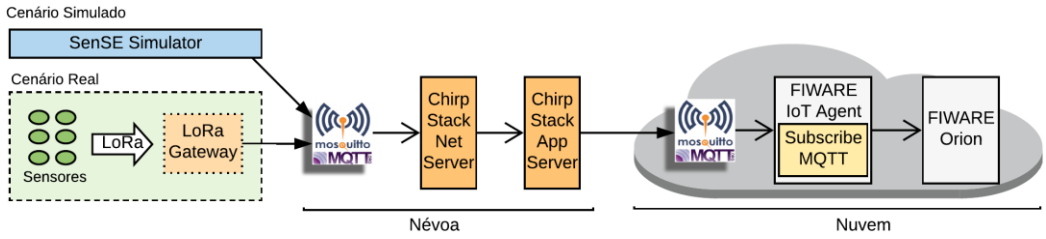


Figura 2. Fluxo de Dados Névoa-Nuvem.

Portanto, para o ambiente de avaliação simulado foram utilizadas duas máquinas (onde a primeira corresponde a névoa e a segunda à nuvem). A névoa computacional possui as seguintes configurações: o sistema operacional Linux Ubuntu 18.04, com processador Intel i5-8265U de 1.60GHz e memória RAM de 8GB. Já a nuvem possui: o sistema operacional Linux Ubuntu 18.04, com 4 núcleos de 2.4GHz de CPU e memória RAM de 8GB. Conectadas por uma rede, com o RTT médio entre as máquinas de 5,67ms ± 3,3ms.

[10] https://www.fiware.org/developers/

Nas avaliações, o simulador SenSE foi responsável por gerar os dados no formato do LoRa Gateway e publicá-los (via MQTT) em um *broker* na névoa, de onde os dados seguem o fluxo pelo ChirpStack Network Server e ChirpStack App Server. Mais tarde, os dados são publicados da névoa para a nuvem, por meio das soluções avaliadas e recebidos pelo IoT Agent (na nuvem).

5.4. Configuração dos Fluxos entre a Névoa e a Nuvem

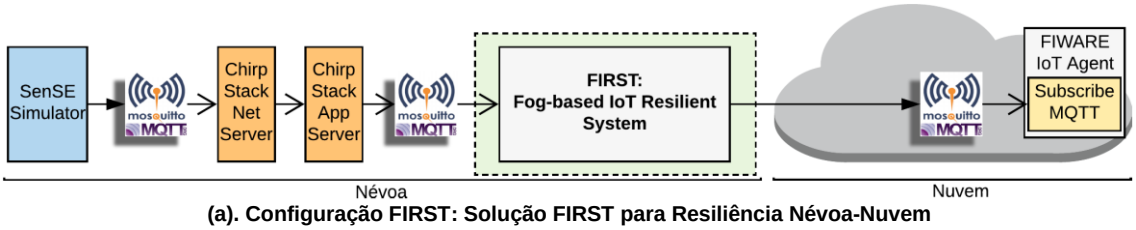
Dentre as configurações para o fluxo de dados entre a névoa e a nuvem, foram avaliadas 4 configurações de fluxos (Figura 3), em que, após o ChirpStack App Server:

a) Os dados são enviados ao MQTT Broker e a solução proposta (FIRST), é responsável por enviar os dados da névoa para a nuvem (Figura 3a);

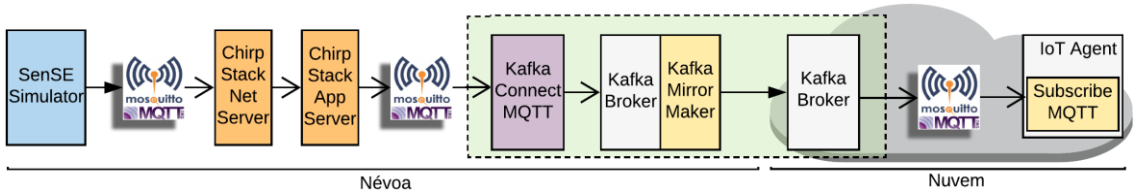
b) Os dados do MQTT Broker (da névoa) são recebidos pelo Kafka Connect MQTT e enviados ao Kafka Broker (na névoa), para que mais tarde sejam enviados (por espelhamento), via Kafka Mirror Maker a um Kafka Broker na nuvem. Dessa forma, na nuvem, os dados seriam consumidos por um IoT Agent (Figura 3b);

c) Os dados são escutados do MQTT Broker (da névoa) pelo (componente) consumidor MQTT do Apache NiFi e enviados à nuvem, pelo (componente) publicador MQTT, do Apache NiFi (Figura 3c);

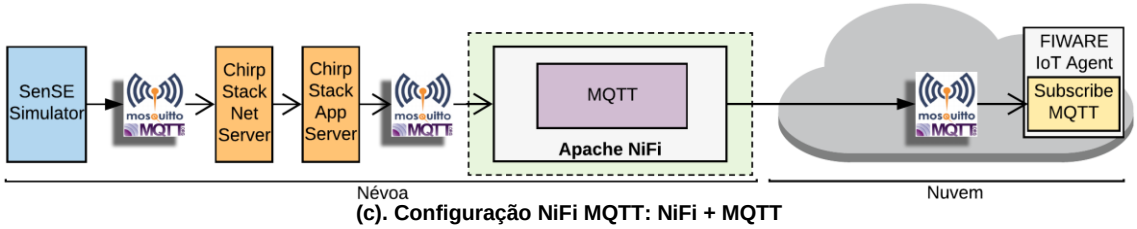
d) Os dados são escutados do MQTT Broker (da névoa) pelo (componente) consumidor MQTT do Apache NiFi e enviados ao Kafka Broker (na névoa), pelo (componente) publicador Kafka, do Apache NiFi. Mais tarde, os dados são espelhados pelo Kafka Mirror Maker (da névoa), para o Kafka Broker (na nuvem), e finalmente escutados do Kafka Broker (nuvem) pelo (componente) consumidor Kafka (do Apache NiFi), e então convertidos ao MQTT Broker (na nuvem), pelo (componente) publicador MQTT do Apache NiFi (Figura 3d).



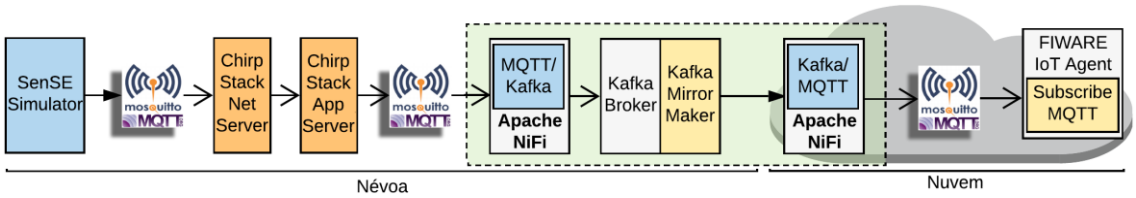
(a). Configuração FIRST: Solução FIRST para Resiliência Névoa-Nuvem



(b). Configuração Kafka: Kafka Connect + Kafka Server + Kafka Mirror Maker



(c). Configuração NiFi MQTT: NiFi + MQTT



(d). Configuração NiFi Kafka: NiFi MQTT + NiFi Kafka + Kafka Server + Kafka Mirror + NiFi Kafka + NiFi MQTT

Figura 3. Configurações dos Fluxos Avaliados.

5.5. Experimentos

Os experimentos foram realizados (e cada um deles replicados 30 vezes), num ambiente análogo ao de uma fazenda inteligente, em dois cenários: (i) em um cenário de disponibilidade de conexão da névoa com a nuvem (em condições normais de operação), cujo simulador SenSE é responsável por enviar os dados à névoa, durante 60 segundos, com uma carga simulada de 1.000, 3.000 e 5.000 sensores; e (ii) em um cenário onde os dados são enviados à névoa (pelo SenSE), durante 60 segundos, porém, em uma situação de desconexão de rede, onde, somente ao final dos 60 segundos, é restabelecida a conexão da rede entre a névoa e a nuvem, numa carga simulada de 1.000 e 5.000 sensores.

6. Resultados

Após executados os experimentos foram obtidos os resultados de cada cenário. 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%. Ademais, foi possível perceber que, assim como observado por [Zyrianoff et al., 2019], a arquitetura LoRa incrementou aproximadamente 200ms de atraso (latência) em todas as configurações avaliadas pelo experimento.

6.1. Cenário de Disponibilidade na Comunicação Névoa-Nuvem

Com relação ao atraso (latência) na chegada dos dados à nuvem (Figura 4), pode se perceber que as configurações *FIRST* e *Kafka* (considerando a margem de erro, para um nível de confiança de 95%) ficaram tecnicamente empatadas, para todas as cargas simuladas, além de terem apresentado, em média, um atraso menor que as demais configurações. Também foi possível perceber que, quando houve um aumento no número de sensores, em média, não houveram variações significativas de latência nas configurações *FIRST*, *Kafka* e *NiFi Kafka*.

No que se refere ao uso de memória RAM da névoa (Figura 5a), dentre as configurações avaliadas, foi possível observar que a configuração *FIRST* demandou menor uso de memória RAM que as demais configurações. Também foi percebido que quando se aumentou o número de sensores, nenhuma das configurações apresentou variações significativas quanto ao uso de memória RAM.

Com relação ao uso de CPU da névoa (Figura 5b), foi percebido que as configurações dos fluxos *NiFi MQTT* e *NiFi Kafka*, que utilizam o Apache NiFi, apresentaram maior uso de CPU que as demais configurações. Além disso, foi percebido que a configuração *FIRST* apresentou, em média, o menor uso de CPU, mesmo quando aumentada a quantidade de sensores.

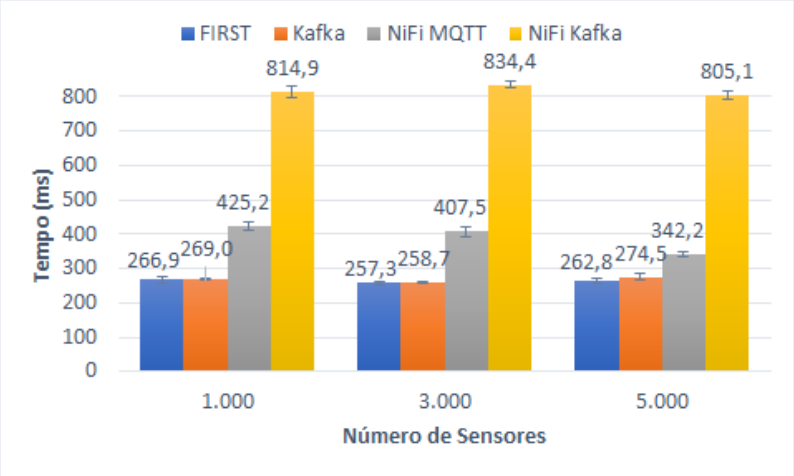


Figura 4. Atraso para diferentes configurações de comunicação Névoa-Nuvem em condições normais de operação.

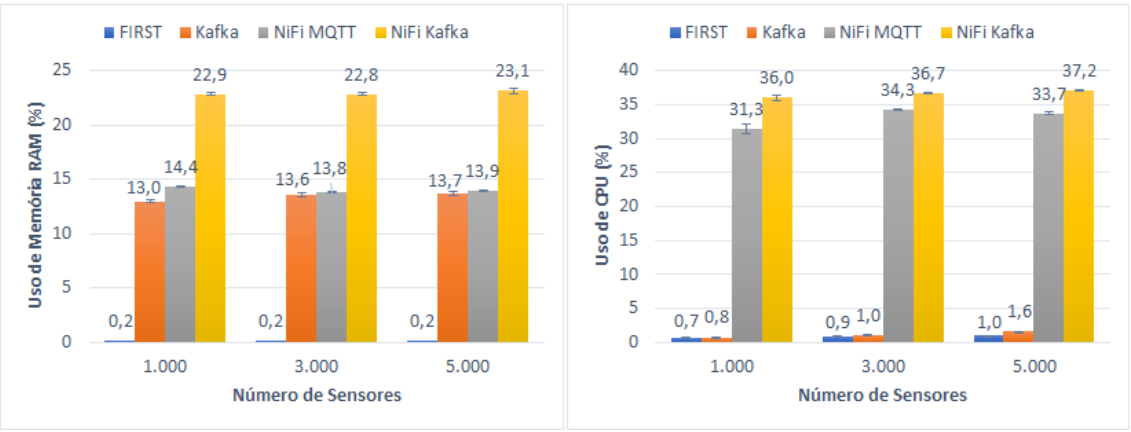


Figura 5. (a) Uso de memória RAM e (b) uso de CPU para diferentes configurações de comunicação Névoa-Nuvem em condições normais de operação.

Foi observado que, à medida que se aumentou o número de sensores no experimento, elevou-se a demanda pelo uso de processamento da névoa, pelas configurações *FIRST*, *Kafka* e *NiFi Kafka*. Também foi observado que, nesse cenário de disponibilidade constante da rede, nenhum pacote foi perdido em nenhuma das configurações avaliadas.

6.2. Desconexão com retomada da Conexão

Para o cenário de desconexão (de duração de 60 segundos), numa carga de 1.000 e 5.000 sensores simulados, foram obtidos os resultados do tempo total de envio de todos os pacotes à nuvem, desde a desconexão (Figura 6a), do tempo de retomada da conexão, desde a desconexão (Figura 6b) e do tempo de envio de todos os pacotes à nuvem, a partir da retomada da conexão (Figura 6c).

Com relação ao tempo total de envio de todos os pacotes à nuvem (Figura 6a), para uma carga de 1.000 sensores, a configuração *FIRST* apresentou 68,07s ± 236,88ms e a configuração *Kafka* apresentou 68,82s ± 294,73ms. Logo, com 1.000 sensores, FIRST apresentou menor tempo de envio (desde a desconexão), isto porque, apesar da configuração FIRST demandar mais tempo para o envio dos pacotes a partir da retomada

da conexão (Figura 6c), o tempo de retomada (Figura 6b) do FIRST foi de 67,58s ± 243,64ms, inferior ao do Kafka de 68,67s ± 293,74ms.

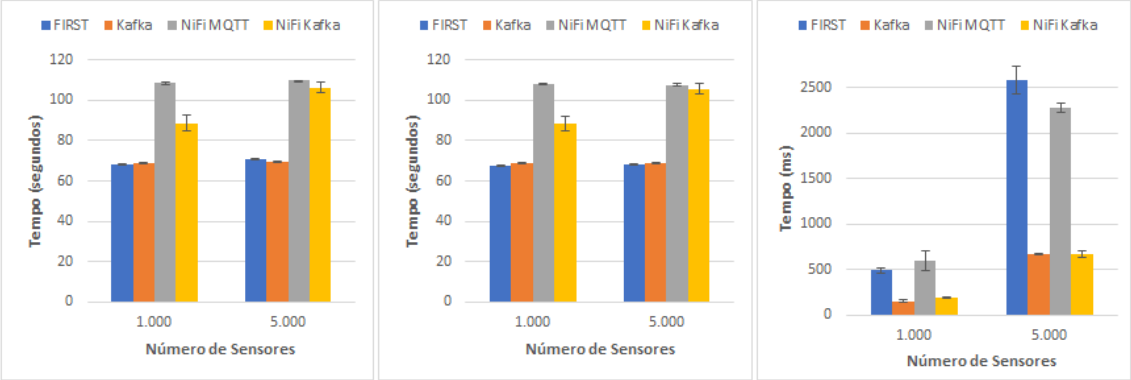


Figura 6. (a) Tempo total de envio de todos os pacotes à nuvem (desde a desconexão); (b) Tempo de retomada da conexão (desde a desconexão); (c) Tempo de envio de todos os pacotes à nuvem, a partir da retomada da conexão (escala diferente).

Contudo, para uma carga de 5.000 sensores, apesar da retomada mais lenta pela configuração *Kafka,* contra *FIRST* (Figura 6b), a configuração *Kafka* apresentou o menor tempo total de envio de todos os pacotes, desde a desconexão (Figura 6a), com valor de 69,59s ± 173,52ms, sendo este tempo inferior ao da configuração *FIRST* com valor de 70,65s ± 264,23ms, isto porque, a configuração *Kafka* envia mais rapidamente todos os pacotes à nuvem, assim que retomada a conexão (Figura 6c).

Foi observado que ambas as configurações que utilizaram o espelhamento do Apache Kafka apresentaram menor valor de tempo para o envio de todos os pacotes, a partir da retomada da conexão (Figura 6c). Ademais, também foi percebido que, mesmo para esse cenário de desconexão, em nenhuma configuração houve perda ou duplicação de pacotes na comunicação entre a névoa e a nuvem.

7. Discussão

Após analisados os resultados, foi possível descobrir que, apesar do Apache NiFi ser uma plataforma facilitadora para a implantação de fluxos de dados, ele demanda um uso de CPU na névoa muito superior ao das demais configurações. Também foi observado que ambas as configurações que utilizaram o Apache NiFi, no cenário de desconexão, apresentaram um atraso significativo no tempo de envio de todos os pacotes (desde a desconexão). Por isso, os tempos de envio de todos os pacotes à nuvem, no cenário de desconexão se apresentaram muito superiores às demais configurações.

Foi percebido que, para nenhuma das configurações, em nenhum dos cenários, apresentou perdas ou duplicação de pacotes. Logo, todas as configurações avaliadas mantiveram a continuidade dos dados no caminho névoa-nuvem nos dois cenários avaliados. Contudo, foi observado em testes preliminares que, quando há uma desconexão entre a névoa e a nuvem, e em seguida, o desligamento e/ou reinicialização na máquina da névoa (para a configuração padrão de instalação do Apache NiFi e do Apache Kafka), somente a configuração *FIRST* foi capaz de retomar a continuidade no fluxo dos dados (que foram recebidos pela névoa), durante o período de desconexão com a nuvem (antes do desligamento da máquina).

Foi constatado que, apesar da configuração *Kafka* ter apresentado, em média, menor tempo total do envio de todos os pacotes à nuvem (desde a desconexão), para uma

carga simulada de 5.000 sensores, esse tipo de abordagem possui certas restrições quando combinada com o fluxo de dados da plataforma FIWARE, já que o FIWARE IoT Agent não suporta mensagens do Apache Kafka. Neste caso faz se necessária a implementação de um agente capaz de compreender o padrão de mensagens do Apache Kafka, para a plataforma FIWARE.

Foi observado que as configurações *FIRST* e *Kafka* apresentaram menor consumo de CPU na névoa, isto porque, para o cenário *Kafka* foi utilizada a menor configuração possível do tópico Kafka (com fator de replicação 1 e um único tópico). Também foi possível observar que a diferença (nas médias do uso de CPU) entre as configurações *FIRST* e *Kafka* aumentou para uma carga simulada de 5.000 sensores.

Percebeu-se que, apesar da solução FIRST, utilizar o MQTT e de ter apresentado melhores resultados com relação ao uso de memória RAM e CPU da névoa, o MQTT Broker não possui mecanismos para lidar com fluxo de dados em *batch* e em tempo real, já que por exemplo, num cenário de desconexão de longa duração, os dados seriam enviados em fila, podendo comprometer o fluxo de dados para análise em tempo real (que aguardariam um longo tempo de atraso na fila).

É válido ressaltar que foram descartadas outras duas configurações de fluxos, pois elas dependeriam do acesso da nuvem ao IP público da névoa (inacessível por conta do NAT). Portanto, configurações que na prática, seriam demasiado incomuns, e somente possíveis caso a névoa tivesse um endereço IP público. Contudo, se fosse possível a implantação: na primeira configuração, o *subscribe* MQTT estaria na nuvem, escutando dados do MQTT Broker da névoa. Já para a segunda configuração, o Kafka Mirror Maker estaria na nuvem, consumindo as mensagens do Kafka Broker da névoa.

8. Conclusão

Este trabalho apresentou uma solução que atendeu aos requisitos de resiliência de dados na comunicação entre a névoa computacional e a nuvem, respeitando as limitações dos recursos computacionais da névoa. Além disso, no trabalho foi realizada uma avaliação da solução proposta, intitulada FIRST, contra outras tecnologias.

Após a avaliação, foi possível perceber que a solução proposta, FIRST, ficou tecnicamente empatada com a configuração *Kafka* em relação à métrica de latência (para condições normais de operação), onde foi observado que as configurações *FIRST* e *Kafka* demandaram menor tempo de latência. Além disso, a proposta FIRST apresentou evidências em demandar menor uso de CPU e memória RAM da névoa, que os demais configurações avaliadas. Portanto, vantajosa em relação às demais configurações de fluxos avaliadas, já que a névoa na maioria dos contextos, possui recursos computacionais limitados.

Pela investigação também foi possível observar que, a proposta FIRST apresentou menor tempo de retomada da conexão que os demais fluxos avaliados. Porém, foi observado que no momento imediato em que é retomada a conexão, o tempo de envio de todos os pacotes à nuvem foi menor para as configurações que utilizaram o Apache Kafka.

Como trabalho futuro, espera-se realizar um estudo com o FIRST, no contexto da bruma computacional, que possui a capacidade de memória RAM e CPU muito mais restrita que a névoa. Também planeja-se aperfeiçoar a proposta, de modo que, através dela, seja possível lidar com dados em *batch* e em tempo real, já que após uma desconexão

de um longo período de tempo, quando restabelecida a conexão da névoa com a nuvem, podem ocorrer atrasos de longa duração, aos pacotes que deveriam ser recebidos em tempo real. Além disso, deseja-se ampliar e aprimorar os requisitos de *trustworthiness* em nível de dados para sistemas baseados em névoa computacional, de modo a contemplar todo o fluxo de dados, partindo dos sensores até a nuvem.

Referências

Aazam M., Zeadally S., and Harras K. A. (2018) "Fog computing architecture, evaluation, and future research directions". IEEE Communications Magazine.

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.

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.

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

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.

Chiang M., Zhang T. (2016) "Fog and iot: An overview of research opportunities". IEEE Internet of Things. Journal, 3(6):854-864.

Dautov R., Distefano S., Bruneo D., Longo F., Merlino G. and Puliafito A. (2018) "Data Processing in Cyber-Physical-Social Systems Through Edge Computing," in *IEEE Access*, vol. 6, pp. 29822-29835, 2018.

Evokly (2016). Kafka Connect MQTT–MIT License. Disponível em: https://github.com/evokly/kafka-connect-mqtt/blob/master/LICENSE. Acesso em setembro de 2019.

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.

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.

Helmer S., Pahl C., Sanin J., Miori L., Brocanelli S., Cardano F., Gadler D., Morandini D., Piccoli A., Salam S., (2016) "Bringing the cloud to rural and remote areas via cloudlets", 7th Annual Symposium on Computing for Development, ACM, p. 14.

Ivaki N., Laranjeiro N., Araujo F. (2018) "A survey on reliable distributed communication", Journal of Systems and Software, Volume 137, Pages 713-732.

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

Kovalchuk O., Gordienko Y. and Stirenko S., (2019) "The Impact of MQTT-based Sensor Network Architecture on Delivery Delay Time," 2019 IEEE 39th (ELNANO*)*,

Kreps J., Narkhede N., and Rao J. (2011) “Kafka: A distributed messaging system for log processing”. In NetDB Workshop, 2011.

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.

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.

Jonathan A., Chandra A. and Weissman J. (2017) “Locality-aware load sharing in mobile cloud computing”, in: Proceedings of the10th International Conference on Utility and Cloud Computing, in: UCC ’17, ACM, pp. 141–150.

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

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.

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.

Machen A., Wang S., Leung K.K., Ko B.J., Salonidis T. (2018) “Live service migration in mobile edge clouds”, IEEE Wirel. Commun. 25 (1) 140–147.

Mouradian C., Naboulsi D., Yangui S., Glitho R. H., Morrow M. J. and Polakos P. A. (2018) “A comprehensive survey on fog computing: State-of-the-art and research challenges”. IEEE Communications Surveys Tutorials, 20(1):416-464.

Mukherjee M., Shu L., and Wang D. (2018) “Survey of fog computing: Fundamental, network applications, and research challenges”. IEEE Communications Surveys Tutorials, 20(3):1826-1857.

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*, vol. 48, no. 7, pp. 37-45.

Ravindra P., Khochare A., Reddy S.P., Sharma S., Varshney P., Simmhan Y. (2017) “ECHO: An Adaptive Orchestration Platform for Hybrid Dataflows across Cloud and Edge”. ICSOC 2017. Lecture Notes in Computer Science, vol 10601. Springer.

Ray P.P., (2018) “A survey on Internet of Things architectures”, Journal of King Saud University - Computer and Information Sciences, Volume 30, Issue 3, Pages 291-319.

Rimal B.P., Van D.P., Maier M. (2017) “Mobile-edge computing versus centralized cloud computing over a converged fiwi access network”, IEEE Trans. Netw. Serv. Manag.

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, Ivan & Heideker, Alexandre & Ottolini, Dener & Kleinschmidt, João & Kamienski, Carlos. (2019). Impacto de LoRaWAN no Desempenho de Plataformas de IoT baseadas em Nuvem e Névoa Computacional. Simpósio Brasileiro de Redes e Sistemas Distribuídos (SBRC 2019).