Composição de Gateways IoT Redundantes com Base em Computação de Nevoeiro

Paulo Thiago G. Mascarenhas[1], Francisco Lopes de Caldas Filho[1], Cassio Fabius C. Ribeiro[1], Fabio L L Mendonca,[1] Rafael T. de Sousa Jr.[1]

***Resumo*— Redes IoT são heterogêneas no que se refere aos dispositivos presentes, incluindo assim dispositivos sem capacidade de se comunicar com a pilha integral de protocolos TCP/IP. Para permitir que sensores e atuadores dessa classe enviem dados através da Internet, é necessário a tais dispositivos se servirem de protocolos como LoRA, ZigBee, Bluetooth, para utilizar o denominado *gateway* IoT, que tem capacidade de roteamento IP. Nesse cenário, há o risco de o *gateway* IoT se tornar um ponto único de falha, podendo deixar sem comunicação todos os dispositivos que dependem dele. Este trabalho propõe a redundância de banco de dados em uma composição de *gateways* IoT sobre rede ZigBee, em configuração de alta disponibilidade sob o comando de um orquestrador em nuvem, de modo a distribuir em modo de computação em nevoeiro, as tarefas computacionais de armazenamento e processamento entre os *gateways* IoT participantes da composição proposta.**

***Palavras-Chave*— Internet das Coisas, gestão de dispositivos, redundância, *gateway IoT*, computação em névoa.**

***Abstract*— IoT networks are heterogeneous with respect to the devices present, thus including devices that are unable to communicate with the full TCP/IP protocol stack. In order to allow sensors and actuators of this class to send data over the Internet, it is necessary to use the so-called IoT gateway, which interconnects devices enabled with protocols such as LoRA, ZigBee, Bluetooth, which generally lack the capability of IP routing. In this scenario, there is a risk that the IoT gateway will become a single point of failure, leaving all elements that depend on it isolated and without communication. This work proposes a database redundancy in a composition of IoT gateways over ZigBee, which operate in high availability and are commanded by a cloud orchestrator, in order to distribute the computational tasks of storage and processing among the remote IoT gateways operating in fog computing architecture.**

***Keywords*— Internet of Things, device management, redundancy, IoT gateway, fog computing.**

I. Introdução

Neste momento, a Internet das Coisas (IoT) está em seu auge. De acordo com [10], o número de dispositivos IoT conectados no mundo ultrapassou a quantidade de pessoas existentes em algum momento de 2017. Outras pesquisas afirmam que esse número era muito superior no mesmo ano, chegando a 27 bilhões [13]. E, por conta da popularização de novos equipamentos inteligentes e do desenvolvimento de novas tecnologias o número de dispositivos IoT está crescendo exponencialmente, conforme dados apresentados em [21].

[1]Departamento de Engenharia Elétrica, Universidade de Brasília (UnB), E-mails: {paulo.mascarenhas, cassio.fabius, francisco.lopes, fabio.mendonça} @uiot.org, desousa@unb.br.

Em ambientes domésticos ou industriais repletos de dispositivos IoT que utilizam protocolos de comunicação heterogêneos para se comunicar pela rede, a presença de um *gateway* IoT é essencial [7]. Por ser um elemento com tanta importância, mas geralmente único, ele acaba se tornando um ponto vulnerável em caso de falha, principalmente na indústria, onde dados costumam ser críticos e mesmo pequenas perdas podem impactar cadeias de produção, análises e monitoramentos, sendo capazes de gerar grandes prejuízos econômicos.

Uma possível solução para esse problema é criar uma redundância nesse ponto, o que proveria alta disponibilidade e a possibilidade de realizar manutenções e melhorias no sistema de maneira a provocar o mínimo possível de prejuízo e perda de dados. Inserir uma redundância física costuma ser caro, mas se o sistema for sensível a períodos de indisponibilidade, o investimento pode valer a pena [15].

Infelizmente a redundância cria alguns inconvenientes: em uma configuração de redundância ativo / passivo, o dispositivo principal processa todas as requisições e o dispositivo backup fica monitorando a saúde do principal para que possa assumir o seu lugar em caso de falha. Assim sendo, os recursos computacionais do dispositivo redundante tendem a ser subutilizados na maior parte do tempo, gastando assim energia desnecessariamente enquanto permanece em espera. Para resolver, ou ao menos minimizar, esse novo revés, há outros recursos que podem ser implementados, como modos de suspensão de energia ou balanceamento de carga, montando uma topologia ativo / ativo.

Uma arquitetura baseada em *Fog Computing* é geralmente melhor que *Cloud Computing* para ambientes IoT [16]. Dessa forma, este trabalho tem como objetivo principal o desenvolvimento de uma solução para replicação de banco de dados em uma arquitetura baseada em *Fog Computing* para *gateway* IoT redundante, reduzindo a dependência da nuvem. A solução também tem foco no melhor aproveitamento dos recursos computacionais que os *gateways* IoT podem oferecer.

A divisão deste artigo é feita da seguinte maneira: a seção 2 mostra trabalhos relacionados, a seção 3 propõe a implementação do *gateway* IoT redundânte com *Fog Computing*, a seção 4 mostra a elaboração de um algoritmo para replicação do banco de dados e a seção 5 descreve as simulações e os resultados experimentais obtidos com o *gateway* proposto. Por fim, a seção 6 mostra as conclusões obtidas com a elaboração do *gateway*, bem como alguns trabalhos futuros.

II. Trabalhos relacionados

O artigo [6] mostra a importância do desenvolvimento da interoperabilidade de protocolos em um *gateway* IoT, que é a base da comunicação entre diferentes tecnologias capazes de se adaptar melhor em cada cenário e aplicação. Tendo em vista essa característica, os autores do trabalho [11] fazem uma abordagem tratando o *gateway* IoT como uma ponte entre a Internet e redes de sensores sem fio. Um *gateway* IoT funcional foi o resultado de sua pesquisa, este é capaz de se comunicar com diferentes protocolos e aplicações. Contudo, ao final do artigo os autores mencionam que critérios de segurança e de tolerância de falhas ainda devem ser adicionados. Este último tópico é o foco do *gateway* que será apresentado neste trabalho, com armazenamento distribuído e o uso de características de *Fog Computing* para agregar mais funcionalidades ao sistema.

A eficiência do uso de *Fog Computing* em um sistema é exemplificada pelo trabalho [2]. Nele os autores mostram dados da implementação de um *gateway* que faz uso de *Fog Computing* para melhorar parâmetros como a utilização núcleo da rede para tarefas desnecessárias. Ao final é possível ver que ainda pode-se explorar a questão de como fazer uma replicação dos bancos de dados no *cluster* de *gateways* IoT.

Outro trabalho que mostra o uso de *Fog Computing*, para diminuir a necessidade de um processamento central dos dados recebidos pelos dispositivos IoT é feito em [20]. Os autores da pesquisa fazem uma modelagem matemática sobre a redundância de *gateways* IoT e falam sobre o uso de tecnologias novas como *Fog Phones* para garantir a alta disponibilidade dos *gateways*. O autor entretanto não demonstra como este objetivo pode ser atingido na prática, citando apenas que o armazenamento e processamento é distribuído antes de enviar os dados para a nuvem.

Em [9], é proposta uma arquitetura de *gateway* IoT semântico capaz de se comunicar com os protocolos ZigBee, MQTT, TCP e UDP. A comunicação ocorre por meio de uma padronização que deve ser seguida na troca de mensagens entre os dispositivos e o *gateway* além das que ocorrem entre *gateway* e *middleware*. O trabalho define o funcionamento interno do *gateway* proposto, mas sem se preocupar com mecanismos que garantam alta disponibilidade e tolerância a falhas. Aqui será o utilizado o protocolo VRRP para garantir redundância no nível de rede, além de sistemas de replicação de base de dados para manter os dados idênticos tanto no *gateway* principal quanto no redundante.

Já no trabalho [14], é possível ver uma redundância implementada a nível de camada 2 em uma rede de sensores sem fio (RSSF) Zigbee. O trabalho faz o uso de um algoritmo próprio para lidar com possíveis problemas que um *gateway IoT* possa ter para operar com caminhos duplicados. O trabalho explora o uso da redundância e como ela pode garantir alta disponibilidade em uma RSSF. Foi utilizado nesta solução o VRRP como protocolo *First Hop Redundancy Protocol* (FHRP). Não há preocupação entretanto com o sincronismo e replicação de base de dados, que neste trabalho é garantido a partir de *scripts* inseridos no dois *gateways* .

III. Proposta: Gateway IoT Redundante com uso de Fog Computing

Redes IoT são compostas por dispositivos que utilizam os mais diversos protocolos e tecnologias de comunicação, como HTTP, MQTT, *Bluetooth*, *Zigbee*, entre outros. Para que todos os dados enviados por esses clientes sejam armazenados num *middleware*, o *gateway* IoT precisa se comunicar através de todos esses protocolos a fim de recebê-los e repassá-los [4]. Entretanto, se uma falha ocorrer nele, todos os dispositivos não terão mais como se comunicar com o *middleware* IoT.

O objetivo desse trabalho é propor modificações no *gateway* IoT redundante desenvolvido em [18] para prover alta disponibilidade e replicação do banco de dados entre os dispositivos que estão na borda da rede. Concomitantemente a isso, fazer com que parte da inteligência da nuvem seja estendida para mais perto dos dispositivos IoT, o que permite reações mais rápidas aos dados recebidos do ambiente; aproveitar de maneira mais eficiente os recursos ociosos disponibilizados após inserção da redundância; e garantir certo nível de independência da Internet.

A arquitetura desenvolvida nesse trabalho (Figura 1) define os dispositivos como sendo responsáveis pela coleta de dados de equipamentos e do ambiente, fazendo parte da *Mist Computing*. Os *gateways* IoT redundantes estão na borda da rede com um *middleware* IoT em execução, aplicando *Fog Computing* ao receber, armazenar, repassar e processar os dados dos dispositivos. Com a redundância, há alta disponibilidade de distribuição das funções do *middleware* entre os *gateways*. Por fim, há um *middleware* em execução na nuvem, também (*Cloud Computing*), onde uma quantidade maior de dados pode ser processada, mas com uma latência maior na comunicação com os dispositivos.

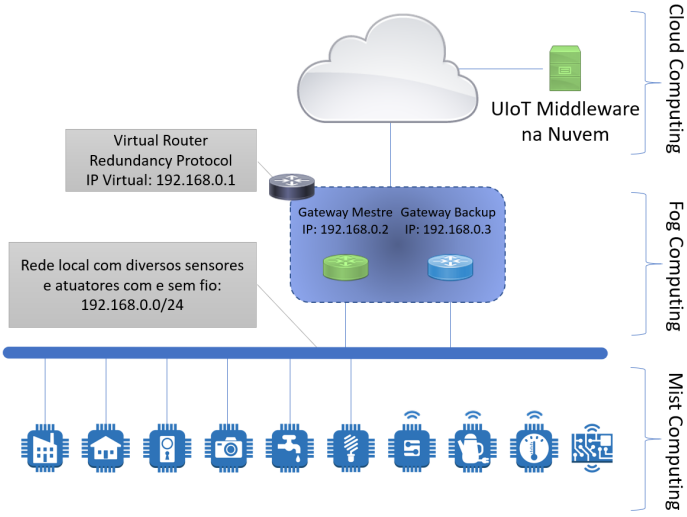

Fig. 1. Arquitetura proposta, mostrando as camadas de Cloud (com o Middleware em Núvem), Fog (onde ficam os gateways) e Mist (onde os diferentes sensores e atuadores aparecem).

Os 2 *gateways* possuem, entre outros, os seguintes *softwares* instalados, configurados e em execução desde a inicialização do sistema operacional para que a proposta ofereça redundância e *Fog Computing*: *UIoT Middleware*, *MongoDB* e

Keepalived.

O ***UIoT Middleware*** é um conjunto de *softwares* que englobam tanto a função de *gateway* IoT quanto de processamento de dados. De acordo com [8], o *UIoT Middleware* tem considerações sobre escalabilidade, transparência, diversidade, mobilidade, performance, facilidade de uso, expansibilidade e é desenvolvido de maneira *open source* e com tecnologias conhecidas. O *UIoT Middleware* será responsável por aplicar a arquitetura de *Fog Computing* na rede IoT. Com essas funções, será possível:

1) Ter suporte aos protocolos HTTP, Socket TCP, Socket UDP, MQTT e *Zigbee*.[4]
2) Atender as requisições feitas pelos dispositivos IoT a fim de recolher dados sobre o ambiente, repassar dados para outros *middlewares* e fazer requisições a dispositivos. [17]
3) Armazenar, processar, exibir e classificar dados recebidos pelos dispositivos IoT de uma rede.
4) Realizar o registro de clientes, serviços e dados para criar uma camada de segurança no serviço.[19]
5) Realizar análise de dados e tratar dados sensíveis à latência de maneira mais eficaz, acionando atuadores da rede a partir do *gateway* IoT.
6) Condensar dados enviados pelos dispositivos para economizar banda ao repassar os dados à nuvem. [3]
7) Dar autonomia à rede local, mantendo os serviços e o armazenamento de dados mesmo que não haja conexão com a Internet. [5]

Foi utilizado o software *Keepalived*, responsável por criar um *cluster* com dois ou mais *gateways*, utilizando o protocolo VRRP. [12] cada *gateway* irá ter um IP na mesma rede dos dispositivos IoT e, o *gateway* principal irá responder pelo IP virtual (VIP), que será o *gateway* padrão da rede. A interface que tiver o VIP será denominada de ***Gateway* Mestre**, enquanto que aquela sem o VIP será denominada de ***Gateway Backup***.

Cada placa possui um *hostname* (GW-01 e GW-02) e um endereço IP fixo. Se o *Gateway* Mestre sofrer uma falha e perder conexão com a rede, o *Gateway Backup* assumirá o VIP e passará a ser o *Gateway* Mestre.

Com o *software UIoT Middleware* ouvindo a porta TCP 8000 será utilizada para receber dados via HTTP dos dispositivos finais. Como as requisições serão enviadas para o VIP, o *Gateway* Mestre sempre receberá os dados. Para manter os bancos de dados dos dois *gateways* sendo populados, os dados recebidos pelo mestre serão enviados para o seu próprio *middleware* local, para o *middleware* do *backup* e para o *middleware* hospedado na nuvem, na plataforma do site Heroku.[1]

Desse modo, o *Gateway* Mestre atuará como *gateway* de comunicação. O *Gateway Backup* terá como funções principais realizar o armazenamento redundante, realizar o processamento dos dados novos e exibir os dados do banco pelo *UIoT Middleware*.

IV. Algoritmo de Replicação do Banco de Dados

A solução proposta tem como requisito principal a alta disponibilidade dos *gateways* e a garantia que haverá pouca ou nenhuma perda de dados quando o *gateway* principal apresentar indisponibilidade. Assim sendo, o protocolo VRRP foi implementado para garantir alta disponibilidade a nível de rede e foram desenvolvidos diversos mecanismos para que houvesse sincronismo constante entre os dois *gateways*. Uma replicação dos bancos de dados entre dois *gateways* da rede, que foi implementada com o uso do *MongoDB*. Para tal, foi necessário desenvolver um algoritmo que pudesse adequar-se aos outros módulos *gateway UIoT* e fosse capaz de enviar dados entre *gateways*.

A Figura 2 mostra como foi desenvolvido o algoritmo. Para cenários de testes foram utilizados 2 *gateways*, 1 como mestre e 1 como backup, ou redundante. Cada um destes equipado com todos os módulos necessários do UIoT, além do pacote *Keepalived*, para prover as funcionalidades do VRRP ao *cluster*.

A Figura 2 mostra o ato de possuir ou não um endereço IP Virtual (VIP) em função do tempo, no cluster mestre-backup. Ou seja, quando um está com o VIP, o outro não está e a troca de VIP ocorre em um intervalo de tempo que foi considerado desprezível para a aplicação e testes, por ser inferior a 3 segundos pode ser incluso dentro do intervalo de 5 segundos de recuperação de uma pane, que será detalhado na próxima seção. A troca é feita pelo serviço implementado pelo *Keepalived* em cada um dos *gateways*. Nos momentos indicados como 1, 2, 3 e 4, o mestre perde o VIP, o mestre ganha o VIP, o backup ganha o VIP e o backup perde o VIP, respectivamente. Cada um destes momentos será descrito nas próximas subseções.

Os períodos de comutação (Cp) são o tempo em que o dispositivo fica com ou sem o VIP. Cada Cp pode ter um tempo diferente e em 1 Cp apenas 1 dispositivo possui o VIP, como é possível verificar na Figura 2.

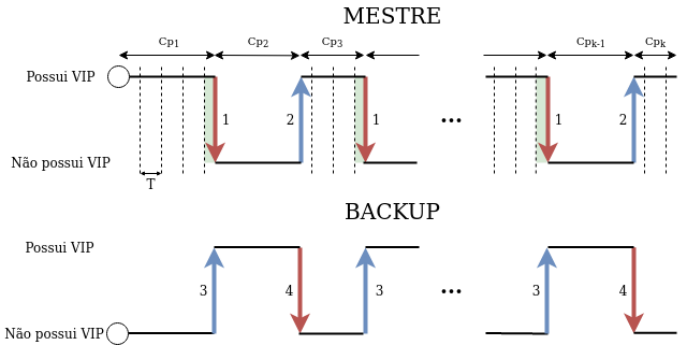

Fig. 2. Algoritmo desenvolvido para replicação do banco de dados

A. Mestre

O mestre da rede executa ações nos momentos 1 e 2, além de ações fixas nos momentos em que se encontram as linhas verticais tracejadas na Figura 2, que aqui serão chamadas de *ticks*. É possível verificar que o código precisa ser executado nos momentos em que o mestre está com o VIP, em um intervalo de tempo fixo entre 2 ticks, denominado T. Para isso, foi utilizado a *crontab* do dispositivo. Além deste momento, também é necessário executar o código nos instantes em que

existe perda ou recepção do VIP. Para isso, foi utilizada a configuração de gatilhos provida pelo *Keepalived*.

Nos instantes representados pelos *ticks*, o mestre deve:

- Verificar em um arquivo auxiliar quantas entradas tinham no banco de dados no momento da execução anterior. Este momento pode ter sido em outro *tick* ou em um instante representado por 2, quando o código é executado.
- Verificar quantas entradas existem no momento atual.
- Ver a diferença entre o momento atual e o anterior.
- Enviar esta diferença e sobrescrever a quantidade de entradas atuais no arquivo auxiliar para que possa ser comparada na próxima execução.

Nos instantes representados por 2, o mestre deve apenas aguardar um tempo, que foi colocado como 5 segundos, para receber possíveis dados que chegaram ao backup enquanto o mestre estava sem conexão. Após o recebimento, ele deve salvar o número atual de entradas no arquivo auxiliar para que possa usar isso na próxima execução em um *tick*.

Por fim, no momento 1, o mestre deve apenas verificar que perdeu o VIP e pode ter mais implementações posteriores.

B. Backup

O backup executa ações nos momentos 3 e 4, sem haver a necessidade de ações fixas ao possuir o VIP pois de nada adiantaria enviar os dados recebidos se o mestre não possui conexão. Desta maneira no instante 3, quando recebe o VIP, o dispositivo deve salvar o número atual de entradas em seu banco de dados e no instante 4, quando perde o VIP, o dispositivo verifica novamente o número de entradas que possui, faz a diferença e a envia para o mestre, que está aguardando por 5 segundos para receber destes dados.

V. Resultados

Para os testes, foi necessário simular situações em que um *gateway* mestre ficava sem conexão e o backup assumia. Para aprimorar os testes e verificar os resultados do algoritmo, foram analisados os valores esperados de perda de pacotes de dados, em comparação com os valores obtidos numa implementação deste algoritmo.

A. Valores esperados de dados perdidos no mestre

Pelo algoritmo, é previsto que o mestre receba dados enquanto estiver com o VIP. Quando este não o possuir, o backup receberá os dados, os armazenará e os transmitirá para o mestre assim que o VIP retorne para ele. Deste modo é esperado que não ocorram perdas de dados no mestre.

B. Valores esperados de dados perdidos no backup

Pelo algoritmo é visto que o backup recebe os dados a cada T segundos, quando o mestre possui o VIP, ou nos instantes em que o backup está com o VIP. Porém dados podem ser perdidos, os que são enviados na região hachurada em verde na Figura 2. Além destes, também podem ser perdidos os dados que chegaram ao mestre de dispositivos que enviaram exatamente nos 5 segundos que ele espera para receber dados após recuperar o VIP, no instante 2, citado anteriormente.

Desta maneira, é possível modelar um valor esperado para os dados perdidos no *gateway* backup. É possível ver que, a cada 2 Cp consecutivos, ocorrem apenas 1 vez os instantes 1, 2, 3 e 4. Isso resulta em um ciclo completo do algoritmo, passando por todas as ações que ele realiza. A cada ciclo destes, ocorrem perdas no backup na região verde (R) e nos 5 segundos após o instante 2 (D_5). Desta maneira, as perdas totais seriam a soma destas duas perdas multiplicada pelo número de vezes que isso ocorre num cenário de testes (Nt). O que seria descrito pela equação abaixo:

$$Perd_{\text{backup}} = (R + D_5).(Nt) \tag{1}$$

O número de dados perdidos na região verde (R) pode ser descrito pelo tempo T entre *ticks*. Considerando o melhor caso, o instante 1 ocorreria imediatamente depois de 1 *tick* e, assim, nenhum dado seria perdido, resultando em R_{min} = 0. Porém, no pior dos casos, isso ocorreria após T segundos, resultando em R_{max}. Como os dados são enviados por um dispositivo a cada Id segundos, constante, temos que R_{max} = T / Id. Assim o valor médio seria de R_m:

$$R_{\text{m}} = \frac{R_{\text{min}} + R_{\text{max}}}{2} = \frac{T}{2Id} \tag{2}$$

Se 1 dado é enviado a cada Id segundos, em 5 segundos. O número de dados que chegam em D_5 será:

$$D_5 = \frac{5}{Id} \tag{3}$$

O valor de Nt dependerá do tempo total do teste e de quantas comutações ocorreram. Como cada período de comutação pode ter um tempo diferente, aqui é preciso considerar a média destes períodos, que será dita Cp_m. Nt será o tempo do teste dividido por 2 Cp_m. Para o tempo total do teste, é preciso saber quantos pacotes são enviados (N) e o intervalo de envio destes pacotes (Id). Com isso é possível obter:

$$Nt = \frac{TempoTeste}{2Cp} = \frac{N.Id}{2Cp_{\text{m}}} \tag{4}$$

Logo, a equação que descreve o número de dados perdidos no backup, baseada na equação 1 é:

$$Perd_{\text{backup}} = \left(\frac{T}{2Id} + \frac{5}{Id}\right).\frac{N.Id}{2Cp_{\text{m}}} = \left(\frac{T}{2} + 5\right).\frac{N}{2Cp_{\text{m}}} \tag{5}$$

Com a equação anterior foi possível verificar então que os dados perdidos no backup dependem do número de dados que são enviados em um teste (N), do intervalo entre os *ticks* (T) e do tempo do médio de comutação Cp_m, independendo então do intervalo entre envio de dados.

Para verificar a aplicação da fórmula para testes reais, foram fixados os parâmetros de número de pacotes, em 1000 e de tempo médio de comutação em 182 segundos. Com isso, a equação que deve descrever os testes fica resumida a:

$$Perd_{\text{backup}} = \left(\frac{T}{2} + 5\right).\frac{1000}{364} \tag{6}$$

O variável T foi colocada em 30, 60 e 120 segundos. Para cada um destes valores, foram feitas 3 baterias de 1000

pacotes enviados e, ao final, uma média para verificar o valor de pacotes perdidos. Assim foi possível obter os resultados apresentados na Tabela I

TABELA I
PACOTES PERDIDOS EM TESTES EM COMPARAÇÃO AO VALOR ESPERADO

Intervalo T	Pacotes perdidos	Valor esperado
30 segundos	52	55
60 segundos	91,33	96
120 segundos	181,00	179

Com os resultados obtidos foi possível ver uma alta precisão da fórmula obtida. Além disso, a implementação do algoritmo possibilitou um aumento considerável na disponibilidade do sistema considerando que, dos 1000 dados enviados, apenas 3, em média, não chegaram ao mestre da rede, o que demonstra uma disponibilidade de 99,7%, ideal para diversos cenários de IoT.

VI. CONCLUSÃO

Este trabalho apresentou uma proposta de um *gateway* redundante IoT com uso de *Fog Computing*, que proveria alta disponibilidade ao *gateway* da rede IoT, garantindo menos interrupções na transmissão de dados dos dispositivos IoT presentes no ambiente, ao mesmo tempo em que oferece a experiência da *Cloud Computing* dentro da própria rede local.

Devido as questões sanitárias atuais, todo o cenário foi implementado por meio de virtualização, porém tudo pode ser replicado em um ambiente real, o que pode interferir na perda de pacotes e na disponibilidade da rede. Mesmo assim, comparando a solução com soluções de apenas 1 *gateway*, esta topologia se sairá melhor.

Como trabalhos futuros, podem ser feitas alterações para que a topologia atue ainda mais como *Fog Computing*, podendo por exemplo fazer *download* de regras que vem de uma entidade central nos *gateways*. Estas regras podem ser desde tempo para medição e envio de resultados até modo de envio. Com elas seria possível melhorar o processamento e enviar apenas o necessário pela internet para um painel de monitoramento e controle centralizado, por exemplo.

Por fim, a topologia implementada pôde prover uma ótima disponibilidade, aumentando o número de dados que chegam ao *gateway* mestre mesmo em uma situação de falha da rede. A solução pode ser útil para diversos ramos como agricultura, indústria e cidades inteligentes, onde pequenas falhas e momentos de indisponibilidade podem ter grandes impactos financeiros.

AGRADECIMENTOS

Os autores agradecem o apoio das Agências brasileiras de pesquisa, desenvolvimento e inovação CNPq (Projetos INCT SegCiber 465741/2014-2, PQ-2 312180/2019-5 e LargEWiN BRICS2017-591), CAPES (Projetos FORTE 23038.007604/2014-69 e PROBRAL 88887.144009/2017-00) e FAPDF (Projetos UIoT 0193.001366/2016 e SSDDC 0193. 001365/2016), bem como o suporte do Laboratório LATITUDE/UnB (Projeto SDN 23106. 099441/2016-43), a cooperação com o Ministério da Economia (TEDs DIPLA 005/2016 e ENAP 083/2016), o Gabinete de Segurança Institucional da Presidência da República (TED 002/2017), a Advocacia-Geral da União (TED 697.935/2019) e o Conselho Administrativo de Defesa Econômica (TED 08700.000047/2019-14).

REFERÊNCIAS

[1] Heroku, cloud application plataform.
[2] M. Aazam and E. Huh. Fog computing and smart gateway based communication for cloud of things. In *2014 International Conference on Future Internet of Things and Cloud*, pages 464–470, 2014.
[3] Francisco Lopes de Caldas Filho. Proposta de um gateway iot em computação fog com técnicas de aceleração wan. 2019.
[4] Francisco L. de Caldas Filho, Martins Lucas M. C., Ingrid P. Araújo, da Costa João Paulo C. L. Mendonça, Fábio L. L., and Rafael T. de Sousa Júnior. Gerenciamento de serviços iot com gateway semântico. In *CIACA, Conferência Ibero Americana de Computação Aplicada*, pages 199–206, 2017.
[5] João TM de Menezes, Pedro HL da Costa, and Dayanne F da Cunha. Desenvolvimento de modelo hierárquico de middlewares com aplicação de fog computing para redes iot.
[6] P. Desai, A. Sheth, and P. Anantharam. Semantic gateway as a service architecture for iot interoperability. In *2015 IEEE International Conference on Mobile Services*, pages 313–319, 2015.
[7] Embitel. What is an iot gateway device and why is it so important for the success of iot projects?, 10 2017.
[8] Hiro Gabriel Cerqueira Ferreira, Caio César de Melo e Silva, Rafael Timóteo de Sousa Junior, and Cláudio Alexander Santoro Wunder. Raise: Rest api approach for iot services.
[9] Francisco L de Caldas Filho, Lucas MC e Martins, Ingrid Palma Araújo, Fábio LL de Mendonça, João Paulo CL da Costa, and Rafael T de Sousa Júnior. Design and evaluation of a semantic gateway prototype for iot networks. In *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 195–201, 2017.
[10] Gartner, Inc. Gartner says 8.4 billion connected "things"will be in use in 2017, up 31 percent from 2016, 2 2017.
[11] S. Guoqiang, C. Yanming, Z. Chao, and Z. Yanxu. Design and implementation of a smart iot gateway. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 720–723, 2013.
[12] IETF. Virtual router redundancy protocol (vrrp) version 3 for ipv4 and ipv6, 3 2010.
[13] IHS Markit. Number of connected iot devices will surge to 125 billion by 2030, ihs markit says, 10 2017.
[14] Paulo Mascarenhas, Francisco Filho, Cássio Ribeiro, Lucas Martins, Pedro Costa, and Rafael de Sousa Junior. Arquitetura de redundÂncia de gateways iot em redes zigbee. In *CIACA, Conferência Ibero Americana de Computação Aplicada*, pages 147–154, 12 2019.
[15] National Instruments Corporation. Redundant system basic concepts, 1 2008.
[16] NWN. Why fog is better than cloud cover in iot, 6 2018.
[17] Rafael L Patrão, Francisco L de Caldas Filho, Lucas MC e Martins, Gerson do N Silva, Matheus S Monteiro, Marcos B Andrade, Fabio LL de Mendonça, and Rafael Timoteo de Sousa Junior. Environmental building monitoring and control based on machine learning and fog computing on an iot architecture. In *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE, 2020.
[18] Cassio Fabius C. Ribeiro, Francisco L. de Caldas Filho, Lucas M. C. e Martins, Cláudia J. Barenco Abbas, and Rafael T. de Sousa Júnior. Protocolos de redundância de gateway aplicados em redes iot. In *XXXVI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais - SBrT2018*, 09 2018.
[19] Caio C. M. Silva, Francisco L. de Caldas, Felipe D. Machado, Fábio L. L. Mendonça, and Rafael T. de Sousa Júnior. Proposta de autoregistro de serviços pelos dispositivos em ambientes de IoT. In *Anais do XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT 2016)*, Santarém, Pará, Brazil, sep 2016.
[20] Soraya Sinche, Oswaldo Polo, Duarte Raposo, Marcelo Femandes, Fernando Boavida, André Rodrigues, Vasco Pereira, and Jorge Sá Silva. Assessing redundancy models for iot reliability. In *2018 IEEE 19th International Symposium on"A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 14–15. IEEE, 2018.
[21] Tim Stack. Data center internet of things (iot) data continues to explode exponentially. who is using that data and how?, 2 2018.