IoTSafe - Uma Arquitetura baseada em Fog Computing para Prover Segurança em IoT

Fabio Coutinho dos Santos[1], **Raquel A. de Freitas Mini**[1]

[1]Departamento de Ciência da Computação – PUC Minas – Belo Horizonte – MG – Brasil

fabio.coutinho@sga.pucminas.br, raquelmini@pucminas.br

Abstract. *The massive use of IoT becomes problematic as it is used in critical systems, as it exposes these systems to various attacks. This paper proposes an architecture based on Fog Computing to provide security in IoT systems. The proposed architecture inserts layers of encryption and authentication between all communications in the system's network, combining encryption protocols such as AES and TLS, with mechanisms such as OAuth2, providing end-to-end security. The results obtained show the effectiveness of the proposal that provides security, being implemented in a Raspberry Pi 3, obtaining latency up to 85 times faster, using up to 5.5 times less CPU compared to a proposal in the literature.*

Resumo. *A utilização massiva de IoT torna-se problemática na medida em que é empregada em sistemas críticos, pois expõe esses sistemas a ataques diversos. Este artigo propõe uma arquitetura baseada em Fog Computing para prover segurança em sistemas IoT. A arquitetura proposta insere camadas de criptografia e autenticação entre todas as comunicações na rede do sistema, aliando protocolos de criptografia como AES e TLS, a mecanismos como OAuth2, provendo segurança de ponta a ponta. Os resultados obtidos evidenciam a eficácia da proposta que fornece segurança, sendo implementada em um Raspberry Pi 3, obtendo latência até 85 vezes mais rápida, utilizando até 5,5 vezes menos CPU em comparação com uma proposta da literatura.*

1. Introdução

A Internet das Coisas (IoT, *Internet of Things*) é um conjunto de sistemas constituídos de dispositivos digitais que se comunicam formando uma rede heterogênea de objetos físicos (prédios, veículos e outros), que são dotados de tecnologia embarcada. Esta rede permite que os objetos se comuniquem e processem dados coletados por meio de sensores, tornando a tomada de decisão em diversos contextos mais inteligente e eficiente [Gupta and Johari 2019]. O crescimento da IoT é exponencial e latente na sociedade contemporânea, tendo atualmente mais de 20 bilhões de dispositivos conectados, com previsão de superar os 30 bilhões em 2024 [Lombardi et al. 2021].

Embora a expansão das aplicações IoT ocasione inúmeros benefícios para a qualidade de vida, o alto número de objetos conectados, trafegando massiva quantidade de dados, expõe cada vez mais os sistemas à ataques. Concomitantemente a essa exposição acentuada, os ataques também se tornam potencialmente mais catastróficos como, por exemplo, ataques a sistemas de transportes, médicos e infraestruturas críticas como usinas de energia [Zarpelão et al. 2017]. O problema de segurança em redes IoT é acentuado

pelo fato de não haver padrões especificamente projetados para dispositivos com recursos computacionais limitados, o que resulta em uma elevada exposição das redes, tornando-as altamente suscetíveis a diversos modalidades de ataques [Frustaci et al. 2017].

Em outubro de 2016, uma grande quantidade dispositivos IoT inseguros ocasionaram uma onda de ataques de *botnet* IoT na infraestrutura da Internet. O *botnet* Mirai comandou 100.000 dispositivos IoT (principalmente câmeras CCTV) para realizar um massivo ataque distribuído de negação de serviço (DDoS) contra a infraestrutura Dyn DNS. Muitos sites populares, incluindo Github, Amazon e Netflix ficaram inacessíveis por diversas horas. Por meio de uma carga estimada de 1,2 Tb por segundo, esse é considerado o maior ataque DDoS já registrado. Em janeiro de 2017, o código-fonte do Mirai foi lançado publicamente, desde então, ataques DDoS usando *botnets* IoT derivados de Mirai aumentaram em frequência e gravidade [Doshi et al. 2018] e [Frustaci et al. 2017].

Duas das principais táticas para reforçar a segurança do tráfego de informações em sistemas IoT são a autenticação dos dispositivos e a criptografia dos dados trocados. Entretanto, as limitações computacionais relativas à capacidade de processamento, memória e armazenamento dos dispositivos periféricos (sensores, atuadores, microcontroladores) tornam soluções clássicas e robustas de criptografia inviáveis para o mundo IoT [Yang et al. 2017] e [McCormack et al. 2020]. Dessa forma, é recorrente na literatura atual a busca por soluções que forneçam privacidade e integridade das informações nos sistemas, com baixo consumo de recursos computacionais e baixa sobrecarga na rede.

Grande parte dessas soluções são baseadas em criação de sessões e compartilhamento chaves públicas, que são compartilhadas pelos elementos do sistema IoT, o que é um problema, pois gera uma sobrecarga de comunicação na rede e, caso essa chave seja descoberta, o sistema é totalmente comprometido. Outro problema das soluções atuais é que elas, geralmente, não são desenvolvidas para o padrão arquitetural emergente de sistemas IoT, focado em computação de borda e redução da latência da rede. Este problema é acentuado na medida em que o emprego de computação de borda utilizando paradigmas como *Fog Computing* se torna o futuro arquitetural dos sistemas IoT [Lin et al. 2017]. Além disso, as propostas atuais, mesmo às que utilizam *Fog Computing*, não fornecem criptografia e autenticação de ponta a ponta do sistema, o que é de extrema importância para a segurança das aplicações [Hassija et al. 2019].

Embora seja o padrão arquitetural emergente, alguns aspectos de *Fog Computing*, como escalabilidade e restrição de recursos computacionais, devem ser cuidadosamente observados. Isso ocorre, pois em comparação com a nuvem, elementos contidos na camada *Fog* podem ter consideravelmente menos poder computacional e menos armazenamento, como também não oferecem escalabilidade sob demanda, uma vez que seus recursos computacionais são limitados. Para mensurar essa diferença, um ambiente em nuvem pode conter poderosos servidores com, por exemplo, 4 núcleos de processamento, 14 GB de memória RAM, 8 discos rígidos e 28 GB de disco SSD [Sohal et al. 2018]. Por outro lado, os *gateways* e elementos contidos na camada *Fog* podem ser constituídos de dispositivos como Raspberry PI e Intel Edison, que possuem capacidade limitada de processamento com apenas 1 GB de memória RAM e 4 GB de armazenamento [Naha et al. 2018].

Diante deste cenário de evolução das arquiteturas dos sistemas IoT, as novas soluções de autenticação e criptografia devem prover segurança de ponta a ponta sendo

viáveis em elementos que possuam poucos recursos computacionais. Essa adequação é necessária, pois permitirá que aplicações que empregam computação de borda e, são considerados o futuro arquitetural dos sistemas IoT, possam utilizá-las. Assim, o objetivo deste trabalho é propor uma arquitetura fundamentada em *Fog Computing*, que contenha autenticação dos dispositivos e criptografia dos dados trafegados em redes IoT, fornecendo integridade e confiabilidade de ponta a ponta para sistema. A arquitetura proposta será focada em baixo consumo de recursos computacionais e baixa latência, empregando protocolos de comunicação otimizados e implementando um *gateway* na borda da rede em um *Raspberry PI 3*, que realizará todo o processamento referente a autenticação dos dispositivos na rede e criptografia dos dados trafegados, contendo apenas 1 GB de RAM DDR2 e um processador Cortex-A53 quad-core de até 1,2 GHz de *clock*.

O restante deste trabalho está disposto da seguinte maneira: Na Seção 2 são explorados os trabalhos relacionados. Na Seção 3 é descrita a solução proposta, abrangendo um aprofundamento acerca de *Fog Computing*, juntamente com a descrição detalhada da arquitetura. A Seção 4 apresenta a avaliação de desempenho da arquitetura, obtida por meio de experimentos práticos realizados. A Seção 5 mostra a conclusão e os trabalhos futuros do estudo. Por fim, todas as referências utilizadas neste trabalho.

2. Trabalhos Relacionados

Propostas de segurança que utilizam autenticação e criptografia em redes IoT geralmente buscam soluções contendo algoritmos que consomem pouco recurso computacional. Esses algoritmos aliados aos protocolos de comunicação, procuram garantir acesso seguro tanto dos usuários aos artefatos das camadas superiores da arquitetura, como servidores e bancos de dados, quanto dos dispositivos periféricos para os elementos de comunicação e transporte da rede. Em geral, essas soluções são formas de mitigar ataques de repetição (reply), *man-in-the-middle* e captura de nó [Hassan et al. 2019].

Nessa tentativa de desenvolver algoritmos leves que forneçam segurança para a rede IoT alguns trabalhos são encontrados na literatura, como [Garg and Dave 2019] e [Razouk et al. 2017]. Nesses estudos, os autores propõem arquiteturas que utilizam *middlewares* de autenticação em uma camada entre os dispositivos periféricos e os elementos de gerenciamento do sistema, ambos utilizando o padrão de comunicação em formato JSON (*JavaScript Object Notation*), por meio de API REST (*Representational State Transfer*). O que difere essas propostas é que em [Garg and Dave 2019] os autores utilizam o padrão OAuth para criptografia, por outro lado, [Razouk et al. 2017] sugerem um protocolo baseado em chaves públicas e funções *Hash*.

Além de soluções arquiteturais que buscam desenvolver algoritmos e esquemas de criptografia próprios, também existem as que combinam tecnologias e protocolos estabelecidos para prover segurança para o ambiente IoT, como sugerem [Castilho et al. 2020]. Nesse estudo, os autores apresentam uma arquitetura que mitiga a possibilidade de ataques em redes IoT, principalmente nos elementos de comunicação e transporte da arquitetura. A proposta consiste em estabelecer a conexão dos elementos que constituem a rede IoT sob a pilha de protocolos IP que são já padronizados e estabelecidos no mercado, como HTTP/HTTPs, TLS/DTLS, TCP, UDP etc. Para tanto, é apresentado um *gateway* com criptografia WPA para implantar segurança na troca de informações.

Outro estudo que apresenta uma arquitetura com *middleware* de segurança en-

tre os dispositivos e as aplicações é [Mukherjee et al. 2017]. Nesse trabalho, os autores argumentam que existe a possibilidade de aplicações IoT em que a velocidade de comunicação pode ser mais importante para a eficácia do sistema que uma estrutura robusta de segurança. Dessa forma, eles sugerem uma solução flexível, em que é possível escolher, dentre algumas opções, o protocolo de transporte, esquema de autenticação, esquema de criptografia e algoritmo de autenticação da mensagem.

Uma proposta que mensura métricas de desempenho computacional e comprova ser uma solução leve, é a apresentada por [Miettinen et al. 2017]. Nesse trabalho os autores apresentam uma arquitetura com uma forma única de autenticar os dispositivos que fazem parte da rede IoT por meio de um *gateway* de segurança. Após identificá-lo, os autores apresentam um método para ranquear esse dispositivo conforme o seu grau de vulnerabilidade, e assim, de acordo com esse ranking, o *gateway* restringe a comunicação deste dispositivo com outros elementos da rede.

Além desses estudos, também há trabalhos como [Alhazmi and Aloufi 2019] e [Sicari et al. 2020] que apresentam soluções baseadas em protocolos projetados e otimizados para IoT, como o MQTT *(Message Queuing Telemetry Transport)* e o COAP (*Constrained Application Protocol*). Em [Alhazmi and Aloufi 2019] os autores apresentam uma proposta que inclui um *web service* para gerenciar os dispositivos que irão estabelecer comunicação por meio do protocolo MQTT. Por outro lado, [Sicari et al. 2020] sugerem uma arquitetura mais completa, baseado em geração e gerenciamento de chaves privadas para prover autenticação entre os dispositivos periféricos e o *broker* MQTT.

Embora as propostas dos estudos mencionados sejam semelhantes à deste artigo, o grande diferencial que ratifica a necessidade deste estudo, é o fato de implementar uma arquitetura que provê autenticação e criptografia de ponta a ponta do sistema aliada a computação de borda, sendo completamente adequada para aplicações de IoT modernas. Isso ocorre pois a arquitetura proposta, denominada *IoTSafe*, irá implementar *Fog Computing* em conjunto com tecnologias atuais, como protocolo de comunicação MQTT, formato JSON e protocolos de criptografia TLS (*Transport Layer Security*) e AES (*Advanced Encryption Standard*), tudo isso para prover mecanismos eficientes de autenticação dos dispositivos na rede e criptografia dos dados, desde a primeira transmissão.

3. Arquitetura Proposta

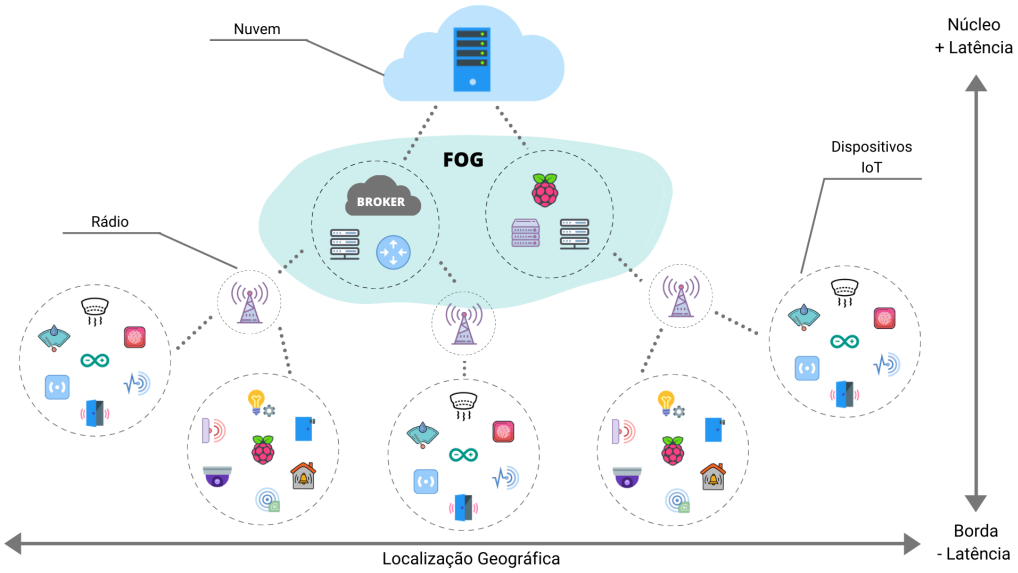
Esta seção apresenta na subseção 3.1 as características e funcionalidades que definem *Fog Computing*, assim como, na subseção 3.2 é descrita detalhadamente a arquitetura proposta denominada *IoTSafe*.

3.1. Fog Computing

Fog Computing é um novo paradigma da computação que consiste em trazer as características e funcionalidades dos serviços de nuvem (*Cloud Computing*) para mais próximo do usuário final em uma aplicação. Devido a essa possibilidade de proximidade do usuário final, a utilização desse conceito em um sistema provê a melhoria da qualidade de serviço QoS (*Quality of Service*) e redução da latência, principalmente naqueles que são projetados para atender densas distribuições geográficas com alta mobilidade [Datta et al. 2015]. Além disso, *Fog Computing* também se torna essencial para o desenvolvimento de sistemas mais eficientes, pois suas características são de suma importância para a eficácia

de aplicações que necessitam de operações em tempo real, com alta confiabilidade e baixa latência, o que torna esse novo modelo o futuro arquitetural do desenvolvimento de aplicações de IoT [Lin et al. 2017]. A Figura 1 ilustra uma arquitetura para sistemas IoT que implementa *Fog Computing*.

Figura 1. Arquitetura IoT com *Fog Computing*



Nesse modelo arquitetural, os dispositivos IoT compostos de sensores e atuadores não possuem comunicação direta com os serviços da nuvem, pois há elementos de processamento e armazenamento, como *gateways* e servidores contidos na camada *Fog*, que estão geograficamente mais próximos deles. Esses elementos da camada *Fog* são responsáveis pelo processamento e armazenamento temporário dos dados, de forma que os serviços que são necessários para realizar esse processamento, que anteriormente ficavam na nuvem, agora são remanejados para a camada *Fog*. Embora tenha sua funcionalidade reduzida, a nuvem ainda é um elemento imprescindível nesse modelo arquitetural, pois ela fica responsável por conter os bancos de dados e os serviços que são necessários para o gerenciamento das aplicações. Essa estratégia faz com que a latência do sistema seja drasticamente reduzida, sendo ideal para aplicações que possuem a necessidade de processamento em tempo real e/ou são projetados para ambientes críticos [O'Donovan et al. 2019].

Em geral, a camada *Fog* é constituída de elementos com pouco poder computacional, entretanto, componentes mais poderosos como computadores pessoais e servidores também podem ser utilizados. Contudo, eles sempre serão limitados e poderão aumentar consideravelmente o custo do sistema, uma vez que nós *Fog* devem ser distribuídos sob diversas regiões geográficas de forma descentralizada. Isso implica que, ao contrário dos sistemas em nuvem que geralmente são constituídos de poucos nós de servidores com vasto recurso computacional, sistemas implementados sob o conceito de *Fog Computing* possuem um grande número de nós, com elementos computacionais de recursos limitados. Dessa forma, é de suma importância que esses elementos que constituem a camada *Fog* sejam de baixo custo para que o sistema seja viável financeiramente, pois o uso deles não dispensa a utilização da nuvem, que ainda é um elemento essencial para gerenciamento

de recursos e armazenamento dos dados. Assim, uma vez que os elementos contidos na camada *Fog* devem ser de baixo custo, automaticamente, isso implica que eles também terão uma capacidade computacional reduzida.

3.2. Arquitetura IoTSafe

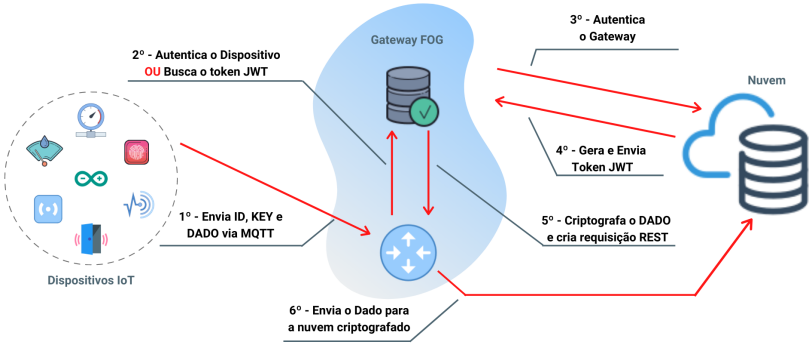
A arquitetura proposta, inicialmente, conta com uma aplicação *front-end* para cadastrar um dispositivo no *gateway Fog* por meio de um número de identificação (ID) único, juntamente com uma chave, para cada dispositivo periférico que utilizará a rede. Para garantir que o ID seja único, ele é gerado no *front-end* utilizando a biblioteca UUID (*Universally Unique Identifier*), que fornece um identificador composto por números hexadecimais, com probabilidade praticamente zero de ser duplicado. Por outro lado, a chave é similar a uma senha que é associado ao ID do dispositivo. Ambos serão gravados em um banco de dados, que também estará contido dentro do *gateway Fog*.

A chave será gravada criptografada por meio da biblioteca *Bcypt*, de maneira que, caso um invasor acesse o banco de dados, ele obterá o ID dos dispositivos, porém não conseguirá descobrir a chave deles, o que impossibilitará este invasor de enviar informações no sistema. Assim como os dispositivos serão cadastrados previamente no *gateway*, o *gateway* também será previamente cadastrado em um banco de dados na nuvem, onde serão gravados os dados do sistema. O processo de cadastro do *gateway* é análogo ao dos dispositivos, por meio de ID e chave. Esse cadastro prévio, tanto dos dispositivos, quanto do *gateway* é necessário para estabelecer uma conexão segura de ponta a ponta do sistema por meio do padrão *OAuth 2*. Toda esta estratégia visa mitigar os ataques que podem decorrer de um invasor conseguir enviar dados na rede, como por exemplo, captura de nó, injeção de dados falsos, acesso ilegal, espionagem entre outros.

Com a chave do dispositivo gravado de forma criptografada no banco de dados, todo o processo de comunicação entre os dispositivos periféricos e o *gateway* ocorre por meio do protocolo MQTT, de forma que a conexão dos dispositivos com o broker ocorre sob criptografia TLS e utilização de autenticação usuário e senha. Por meio deste protocolo, os dados serão enviados para o *gateway* em formato JSON contendo o ID do dispositivo, a chave e o dado. Para exemplificar, se o dispositivo de ID ”3538191c13ad11eba68474d4359afc5a” e chave ”12345” for enviar o dado ”Teste”, o envio deve ser realizado no seguinte formato: {”deviceID”:”3538191c13ad11eba68474d4359afc5a”, ”key”:”12345”,”data”:”Teste”}.

O MQTT foi criado pela empresa IBM (*International Business Machines Corporation*) em 1999 e foi utilizado internamente por ela até o ano de 2010, quando foi lançado como *open source* e padronizado pela OASIS (*Organization for the Advancement of Structured Information Standards*) e ISO (*International Organization for Standartzation*). Assim, após sua padronização, o MQTT se tornou um protocolo amplamente utilizado em ambientes M2M (*Machine-to-Machine*) e IoT, devido a ser orientado a mensagem, possuir baixa sobrecarga e necessitar de baixa largura de banda, o que o credência para ser utilizado de forma eficiente em dispositivos com recursos computacionais limitados [Bideh et al. 2020]. As escolhas pelo protocolo MQTT e o formato de mensagens JSON se dão pelo fato de ambos serem utilizados amplamente no mercado e na literatura, além de serem reconhecidamente padrões eficientes que diminuem a sobrecarga e latência no tráfego de informações da rede [Garg and Dave 2019] e [Bideh et al. 2020]. A Figura 2 ilustra a arquitetura proposta, ilustrando funcionamento do sistema de forma geral.

Figura 2. Arquitetura IoTSafe



Todo o processo de comunicação segura por meio do padrão *OAuth* 2 acontece da seguinte maneira: inicialmente, o *gateway* irá conferir se o dispositivo que está enviando o dado está autenticado no banco de dados do *gateway*. Caso o dispositivo não esteja autenticado, o *gateway* consulta pela *KEY* se o dispositivo está cadastrado e caso esteja, autentica o dispositivo, caso não esteja cadastrado, ele interrompe a comunicação. Em seguida, o *gateway* realiza uma requisição REST para o servidor na nuvem com seu ID e *KEY* para que também seja autenticado. Caso haja a autenticação, o servidor da nuvem devolve para o *gateway* um *token* JWT (*JSON Web Token*) que irá permitir acesso ao banco de dados. Esse *token* tem um período de validade e é gravado no banco de dados do *gateway* relacionado ao ID do dispositivo.

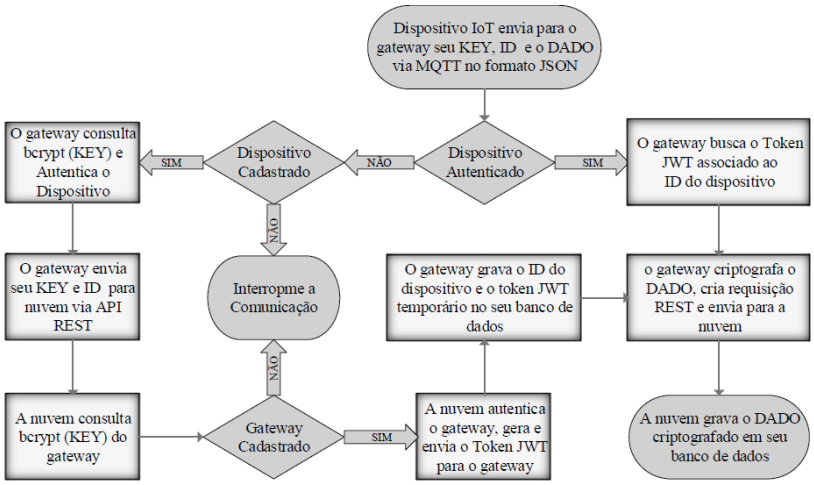
Assim que o *gateway Fog* recebe o *token* é estabelecida uma conexão segura de ponta a ponta entre o dispositivo e a nuvem, possibilitando que o dispositivo envie os dados. O dado enviado após o estabelecimento da conexão segura, será criptografado por meio de um algoritmo AES *(Advanced Encryption Standard)* em que uma *string* chave é utilizada para critografar e descriptografar o dado. Após criptografar o dado recebido, o *gateway Fog* constrói uma requisição REST, contendo a *KEY* e o ID do dispositivo, o dado criptografado e o *token* de autorização no cabeçalho, logo em seguida, essa requisição é enviada para o banco de dados na nuvem, protegida por criptografia TLS, por meio do protocolo HTTPs (*Hypertext Transfer Protocol Secure*). A Figura 3 demonstra de forma mais detalhada como será o fluxo de ações da arquitetura.

O fluxo maior, em que o dispositivo IoT não é autenticado imediatamente no *gateway* ocorre apenas na primeira requisição do dispositivo para o sistema, ou quando o *token* JWT expira, de forma que requisições realizadas após esse fluxo inicial tomará um caminho mais curto, em que dispositivo é autenticado no segundo passo da Figura 2. Assim, após a confirmação da autenticação deste dispositivo, o *gateway Fog* apenas busca em seu próprio banco de dados o *token* relacionado ao ID do dispositivo, eliminando o 3º e 4º passos da Figura 2, e então ele criptografa o dado e cria a requisição REST que será enviada para o banco de dados da nuvem.

4. Avaliação de Desempenho

Esta seção mostra, na subseção 4.1, o ambiente elaborado para realização dos experimentos na arquitetura proposta, assim como descreve detalhadamente cada um dos experimentos aplicados na subseção 4.2. Por outro lado, a subseção 4.3 apresenta um discussão acerca dos resultados obtidos.

Figura 3. Fluxo de Ações de Autenticação e Criptografia



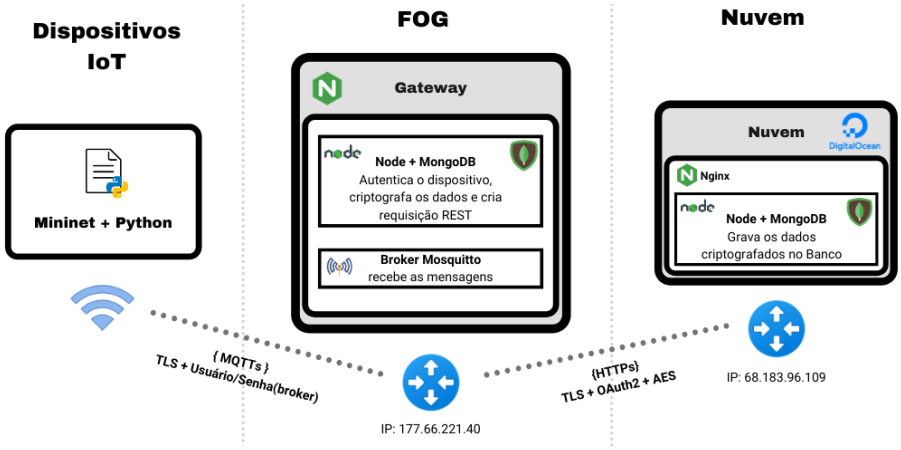
4.1. Ambiente de Teste

Inicialmente foi elaborado uma aplicação *front-end* desenvolvida por meio do *framework* AngularJS para realizar o cadastro tanto dos dispositivos no *gateway*, quanto do *gateway* na nuvem. Além da aplicação *front-end*, também foram elaboradas duas aplicações de *back-end*, sendo uma em um Raspberry Pi 3, e outra em um servidor na nuvem. A aplicação implementada no Raspberry Pi 3 foi o *gateway*, sendo o elemento da camada *Fog* da arquitetura do sistema, responsável por todo o processamento relativo a receber a requisição via MQTT dos dispositivos IoT, autenticar este dispositivo e criptografar os dados. Por outro lado, a aplicação desenvolvida na nuvem é a responsável por manter o banco de dados com os dados enviados pelo *gateway* de forma criptografada, para que possa servir de *back-end* para aplicações que utilizam esses dados.

Para implementar o *gateway* no Raspberry Pi foi necessário a criação de um servidor privado. Para isso foi utilizado o *software* de servidor *web* de código aberto Nginx, além de uma API REST desenvolvida em NodeJs juntamente com um banco de dados Mongo. Em conjunto com a API REST desenvolvida, também foi instalado um broker MQTT Mosquitto, protegido por usuário/senha e criptografia TLS para receber as mensagens enviadas pelos dispositivos IoT. Por outro lado, para criação do *back-end* na nuvem, foi utilizado o serviço do provedor de serviços em nuvem *Digital Ocean*. Por meio deste provedor, foi instalado um *droplet*, ou um ambiente computacional constituído também por um servidor Nginx e uma API REST em conjunto com um banco de dados Mongo, tudo gerenciado por um sistema operacional Ubuntu Server v 20.04.

Além do *gateway Fog* e da aplicação em nuvem, também foi elaborado um script em python, que em conjunto com o simulador de rede Mininet, simulam os dispositivos IoT que utilizam o sistema enviando mensagens para o *gateway* via protocolo MQTT. O Raspberry Pi que hospeda a aplicação *back-end* do *gateway Fog* é gerido por um sistema operacional Raspbian, possui 1 GB de memória RAM e um processador Broadcom Cortex-A53, com 4 núcleos de processamento de 1.2 GHz de *clock*. Por outro lado, a maquina virtual da nuvem na *Digital Ocean* possui 1 GB de memória RAM, 25 GB de armazenamento de dados e apenas 1 núcleo de processamento compartilhado. A Figura 4 ilustra o ambiente completo elaborado, integrando todas as aplicações desenvolvidas.

Figura 4. Ambiente de teste elaborado



Por meio deste ambiente os dispositivo IoT são simulados utilizando o Mininet, em seguida, mensagens contendo dados de teste são enviados para o *gateway Fog* via protocolo MQTTs, o broker possui o mecanismo de autenticação usuário/senha, além de criptografia TLS, ou seja, só estabelece uma conexão por meio de portas criptografadas e com dispositivos que enviem o usuário e a senha do broker preestabelecida . O *gateway* recebe esses dados por meio do *broker* MQTT Mosquitto instalado e realiza o processamento para autenticar o dispositivo que enviou a mensagem, criptografar o dado que foi enviado e criar a requisição REST para encaminhar o dado para a nuvem, juntamente com o ID do dispositivo que o enviou.

A aplicação da camada hospedada na nuvem recebe o dado criptografado por meio de uma API REST e o grava no banco de dados. Tanto no *gateway*, quanto na nuvem, o servidor Nginx é o que permite que as rotas das APIs REST fiquem disponíveis na internet, funcionando como *proxy* reverso, estabelecendo quais rotas ficam disponíveis *on-line* e roteando o tráfego que chegam nelas para as rotas internas da aplicação em NodeJS. Além disso, o Nginx também insere um certificado ssl para as aplicações, tornando-as ainda mais seguras, passando a trafegar as informações sob criptografia TLS.

4.2. Experimentos realizados

Esta subseção apresenta cada teste realizado para avaliar quantitativamente a arquitetura proposta. Para tanto, são demonstrados os testes de consumo de recursos na subseção 4.2.1 e os testes de latência na subseção 4.2.2.

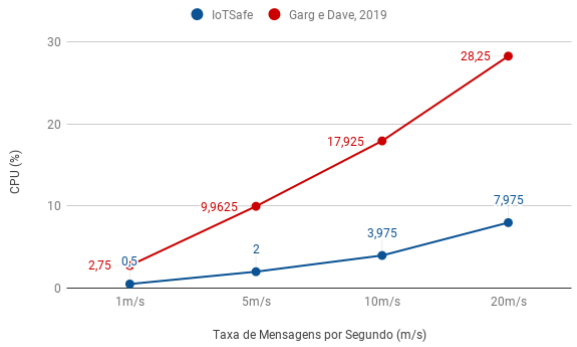
4.2.1. Consumo de Recursos

Este teste tem o objetivo de avaliar o desempenho do *IoTSafe* ao realizar as tarefas de autenticação dos dispositivos e criptografia dos dados. Para isso, foi utilizado um *framework* desenvolvido em python chamado psrecord. Esse *framework* permite que processos do sistema operacional sejam monitorados, fornecendo o quanto de CPU e de memória RAM estes processos utilizaram. Dessa forma, o processo responsável por executar o programa NodeJs que realiza todo processamento das mensagens no *gateway* foi monitorado para captura das métricas.

Para validar os resultados, a arquitetura proposta em [Garg and Dave 2019] também foi implementada e o processo que realiza o processamento das mensagens no *gateway* também foi monitorado para a captura das métricas. A arquitetura proposta por [Garg and Dave 2019] foi escolhida para a comparação de desempenho pois é, das encontradas na literatura, a proposta que mais se assemelha com a *IoTSafe*, apresentando um fluxo completo de ações e mecanismos de segurança desde que a comunicação é enviada pelos dispositivos IoT até que ela chegue no núcleo da rede em ambientes em nuvem. Assim, ela foi implementada conforme a proposta dos autores, estabelecendo uma comunicação via bluetooth com um *middleware* de autenticação, que para a realização desta comparação foi implementado no Raspberry PI 3. Esse *middleware* recebe a mensagen, autentica o dispositivo que a enviou e em seguida a encaminha para o banco de dados da nuvem por meio de uma API REST, utilizando autenticação Oauth2.

Assim, para comparação das propostas, as mensagens dos dispositivos periféricos simulados pelo Mininet, foram enviadas utilizando as frequências de 1, 5, 10 e 20 mensagens por segundo, via MQTTs na IoTSafe e via *bluetooth* na proposta de [Garg and Dave 2019]. Para efetuar a captura das métricas o simulador enviou mensagens em cada uma das frequências utilizadas por 1 minuto, enquanto os processos do *gateway*, citados anteriormente, foram monitorados. A Figura 5 mostra a utilização de CPU da *IoTSafe* e também da proposta de [Garg and Dave 2019].

Figura 5. Utilização de CPU



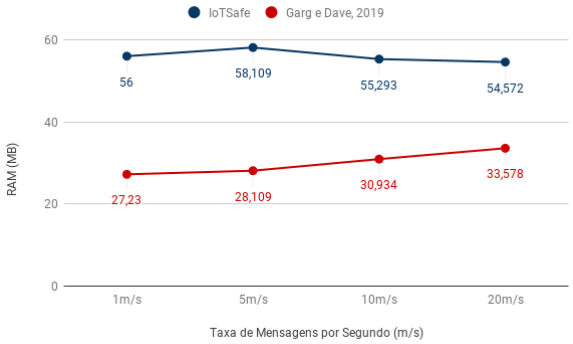
A Figura 5 demonstra graficamente que o gasto de CPU do *gateway* na execução da *IoTSafe* foi consideravelmente menor, em que para a frequência de 1m/s (mensagens por segundo) a arquitetura proposta por [Garg and Dave 2019] utilizou cerca de 5,5 vezes mais CPU e para a frequência de 20m/s a diferença reduziu, mas ainda se manteve em 3,54 vezes. A Figura 6 mostra o consumo de memória RAM de ambas as propostas.

A Figura 6 demonstra uma desvantagem da *IoTSafe* em relação ao consumo de memória RAM, sendo que para a frequência de 1m/s a *IoTSafe* utilizou 2,05 vezes mais memória e na frequência de 20m/s a a diferença diminuiu para 1,62 vezes.

4.2.2. Latência

Para realização deste teste, a mensagem gerada pelo *script* python, que simula os dispositivos IoT, captura o *timestamp* do sistema no momento que o objeto JSON que contém os campos da mensagem é criado. Dessa forma, esse objeto JSON é enviado para o *gateway*

Figura 6. Utilização de memória RAM



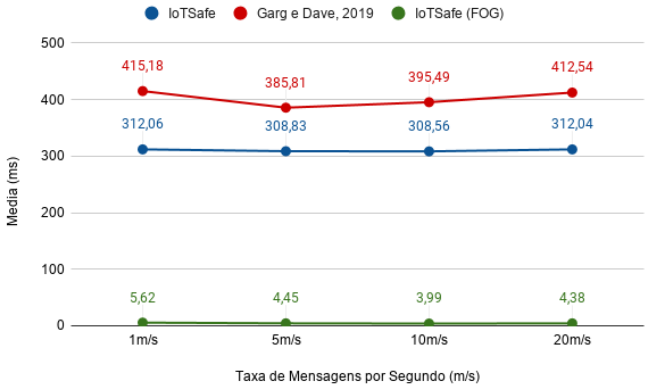
contendo os seguintes campos: *Code, DeviceId, Key, Data e Timestamp*. O campo *Code* é utilizado para identificar sob qual taxa de envio em relação ao tempo a mensagem foi enviada. O *gateway* recebe esta mensagem e imediatamente realiza o processamento para autenticar o dispositivo que a esta enviando e, em seguida, criptografar o dado recebido no campo *Data*. A criptografia ocorre por meio do padrão AES necessitando de uma *string* chave, que, para os testes realizados foi uma sentença formada de números hexadecimais pela biblioteca UUID ”70fd019e13f611ebaae874d4359afc5a”. Assim, após realizar a criptografia do dado e autenticar o dispositivo, a requisição REST é criada com o *token* JWT, referente ao dispositivo que enviou a mensagem, como cabeçalho de autorização. Além disso, na criação dessa requisição são adicionados mais dois campos ao objeto JSON, timestampFinal e tempoTotal.

Após a criação dessa mensagem, o *gateway* a envia para a nuvem para ser gravada no banco de dados, com os campos timestampFinal e tempoTotal contendo uma *string* vazia. Assim, quando essa mensagem chega na aplicação da nuvem, imediatamente o *timestamp* do sistema é capturado novamente e é realizada a subtração deste novo *timestamp* com o anterior, capturado e enviado pelo *script* python no início do processo. Dessa forma, é calculado o tempo gasto entre a mensagem ser enviada pelo dispositivo periférico, ser processada pelo *gateway* e gravada no banco de dados na nuvem.

De forma análoga ao teste de consumo de recursos, a arquitetura proposta por [Garg and Dave 2019] também foi testada, contudo, ao invés de a mensagem ser enviada para o *gateway* por MQTT, foi utilizado *bluetooth*, sendo que o processo de cálculo da latência, assim como o tamanho da mensagem e método de criptografia utilizado foi o mesmo que o descrito acima. Assim, utilizando este processo para cálculo da latência, as mensagens foram enviadas variando a taxa de mensagens por segundo em 1,5,10,20. Além da latência entre a mensagem sair dos dispositivos periféricos e ser gravada no banco de dados da Nuvem, também foi mensurado a latência da mensagem chegar e ser gravada no banco de dados contido na camada *Fog* da *IoTSafe*. A Figura 7 mostra graficamente os resultados obtidos.

A Figura 7 mostra a evidente superioridade da *IoTSafe* em relação a proposta de [Garg and Dave 2019], sendo que para todas as frequências testadas, a *IoTSafe* foi aproximadamente 1,3 vezes mais rápida, considerando a latência até a mensagem chegar ao banco de dados na nuvem. Além disso, uma opção que a *IoTSafe* oferece é gravar os dados em um banco dentro do próprio *gateway*, caso a aplicação esteja geograficamente

Figura 7. Comparação de Latência



próxima ou na mesma rede do *gateway*. Dessa forma, a latência passa a ser apenas o tempo entre a mensagem ser enviada e gravada no *gateway*, uma opção que a arquitetura proposta por [Garg and Dave 2019] não possui. Considerando este cenário, a *IoTSafe* e aproximadamente entre 75 a 85 vezes mais rápida.

4.3. Discussão

Os resultados obtidos por meio da avaliação de desempenho da arquitetura IoT-Safe demonstram que ela é uma evidente evolução da arquitetura proposta por [Garg and Dave 2019], pois além de poder ser utilizada no mesmo contexto tecnológico com mais eficiência, ela também oferece outras funcionalidades devido ao emprego de computação de borda. Além disso a IoTSafe preenche algumas lacunas observadas de outros trabalhos da literatura, como por exemplo em [Razouk et al. 2017]. Nesse estudo os autores inserem o conceito de *Fog Computing* na solução para agilizar o processo de comunicação e diminuir a latência da rede, entretanto, a proposta não fornece criptografia de ponta a ponta para o sistema e não mensura o consumo de recursos.

Por outro lado [Mukherjee et al. 2017] apresentam uma solução que consome pouco recurso computacional. Entretanto, além de comprometer a latência do sistema por não implementar computação de borda na proposta, o que também acontece em [Castilho et al. 2020], eles testam 4 combinações de mecanismos de segurança e criptografia diferentes, em que apenas as duas que possuem menor grau de segurança, baseadas em uma chave de criptografia estática (*PSK*) obtém um consumo de memória próximo ao do *gateway* proposto na IoTSafe, sendo que as outras duas obtém um consumo de ordem de grandeza consideravelmente superior.

Em relação a [Sicari et al. 2020] e [Alhazmi and Aloufi 2019], a primeira sugere a utilização de *Fog Computing* integrando-a ao protocolo MQTT na solução, porém não emprega um modelo de criptografia de ponta a ponta na rede, além de consumir cerca de 4 vezes mais processamento em comparação com a IoTSafe na frequência de 10 m/s. Por outro lado, [Alhazmi and Aloufi 2019] sugerem um *web service* para gerenciar permissões dos dispositivos para se inscreverem em um broker MQTT. Essa estratégia pode aumentar muito a latência do sistema caso este *web service* não esteja implementado na borda da rede, ou seja, caso a utilização de *Fog Computing* seja desprezada. Além disso, os autores também não apresentam um modelo de criptografia de ponta a ponta.

5. Conclusão e Trabalhos Futuros

Devido ao aumento eminente das aplicações de IoT e, consequentemente da quantidade de dados trafegados nas redes desses sistemas, é de suma importância que arquiteturas eficientes e seguras sejam implementadas. Dessa forma, a *IoTSafe* garante o armazenamento seguro das credenciais de acesso nos dois níveis, borda e núcleo da rede. Além disso, barreiras de autenticação e criptografia são empregadas em todos os níveis de comunicação, sendo a autenticação do broker e criptografia TLS entre os dispositivos IoT e o *gateway*, e a criptografias TLS e AES além de autenticação Oauth2 entre o *gateway* e a nuvem, garantindo segurança, privacidade e integridade dos dados de ponta a ponta, tudo isso com baixa latência e implementado em um ambiente com pouco recurso computacional.

Como trabalho futuro, pretende-se acrescentar um módulo de predição de ataques a *IoTSafe*, de forma que, caso os mecanismos de criptografia e autenticação sejam insuficientes, esse módulo de predição seja mais uma barreira do sistema. A implementação desse módulo tornará a arquitetura mais robusta e eficiente, entretanto, manter a baixa latência com pouco recuso computacional será um desafio a ser explorado.

Agradecimentos

Este trabalho teve o apoio da FAPEMIG, CAPES e CNPq.

Referências

Alhazmi, O. H. and Aloufi, K. S. (2019). Fog-based internet of things: a security scheme. In *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–6. IEEE.

Bideh, P. N., Sönnerup, J., and Hell, M. (2020). Energy consumption for securing lightweight iot protocols. In *Proceedings of the 10th International Conference on the Internet of Things*, pages 1–8.

Castilho, S. D., Godoy, E. P., and Salmen, F. (2020). Implementing security and trust in iot/m2m using middleware. In *2020 International Conference on Information Networking (ICOIN)*, pages 726–731. IEEE.

Datta, S. K., Bonnet, C., and Haerri, J. (2015). Fog computing architecture to enable consumer centric internet of things services. pages 1–2.

Doshi, R., Apthorpe, N., and Feamster, N. (2018). Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. IEEE.

Frustaci, M., Pace, P., Aloi, G., and Fortino, G. (2017). Evaluating critical security issues of the iot world: Present and future challenges. *IEEE Internet of things journal*, 5(4):2483–2495.

Garg, H. and Dave, M. (2019). Securing iot devices and securelyconnecting the dots using rest api and middleware. In *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–6. IEEE.

Gupta, A. K. and Johari, R. (2019). IOT based Electrical Device Surveillance and Control System. *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–5.

Hassan, W. H. et al. (2019). Current research on internet of things (iot) security: A survey. *Computer networks*, 148:283–294.

Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., and Sikdar, B. (2019). A survey on iot security: application areas, security threats, and solution architectures. *IEEE Access*, 7:82721–82743.

Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., and Zhao, W. (2017). A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet of Things Journal*, 4(5):1125–1142.

Lombardi, M., Pascale, F., and Santaniello, D. (2021). Internet of things: A general overview between architectures, protocols and applications. *Information*, 12(2):87.

McCormack, M., Vasudevan, A., Liu, G., Echeverría, S., O'Meara, K., Lewis, G., and Sekar, V. (2020). Towards an architecture for trusted edge iot security gateways. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association.

Miettinen, M., Marchal, S., Hafeez, I., Asokan, N., Sadeghi, A.-R., and Tarkoma, S. (2017). Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184. IEEE.

Mukherjee, B., Neupane, R. L., and Calyam, P. (2017). End-to-end iot security middleware for cloud-fog communication. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 151–156. IEEE.

Naha, R. K., Garg, S., Georgakopoulos, D., Jayaraman, P. P., Gao, L., Xiang, Y., and Ranjan, R. (2018). Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE access*, 6:47980–48009.

O'Donovan, P., Gallagher, C., Leahy, K., and O'Sullivan, D. T. (2019). A comparison of fog and cloud computing cyber-physical interfaces for industry 4.0 real-time embedded machine learning engineering applications. *Computers in Industry*, 110:12–35.

Razouk, W., Sgandurra, D., and Sakurai, K. (2017). A new security middleware architecture based on fog computing and cloud to support iot constrained devices. In *Proceedings of the 1st International Conference on Internet of Things and Machine Learning*, pages 1–8.

Sicari, S., Rizzardi, A., and Coen-Porisini, A. (2020). Increasing the pervasiveness of the iot: fog computing coupled with pub&sub and security. In *2020 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pages 64–71. IEEE.

Sohal, A. S., Sandhu, R., Sood, S. K., and Chang, V. (2018). A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments. *Computers & Security*, 74:340–354.

Yang, Y., Wu, L., Yin, G., Li, L., and Zhao, H. (2017). A survey on security and privacy issues in internet-of-things. *IEEE Internet of Things Journal*, 4(5):1250–1258.

Zarpelão, B. B., Miani, R. S., Kawakani, C. T., and de Alvarenga, S. C. (2017). A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications*, 84:25–37.