Análise do Protocolo MQTT para Comunicação IoT através de um Cenário de Comunicação

Wellington Nogueira Elizeu da Conceição, Romualdo Monteiro de Resende Costa

Curso de Bacharelado em Sistemas de Informação –Centro de Ensino Superior de Juiz de Fora(CESJF) – Campus Academia
36016-000 – Juiz de Fora– MG– Brasil

wellingtoncw7@gmail.com, romualdomrc@gmail.com

Abstract. The technological advances that have occurred in the last decade have provided advances in several areas, benefiting professionals and citizens from all over the world. The Internet of Things (IoT) is a scenario where thousands of devices are connected to the Internet, automating and facilitating communication, bringing about improvements in how we interact with objects on a daily basis. To this end, it is necessary to have specific communication protocols that enable the communication of a large number of devices. In this scenario, this work aims to present the MQTT protocol as a solution for communication between IoT clients.

Resumo. Os avanços tecnológicos ocorridos na última década propiciaram avanços em diversas áreas, beneficiando profissionais e cidadãos de todo o mundo. A Internet das Coisas ou *Internet of Things* (IoT), é um cenário onde milhares de dispositivos estão conectados à Internet, automatizando tarefas, facilitando a comunicação e trazendo vantagens na forma como interagimos com objetos no dia a dia. Para que a comunicação seja facilitada, é necessário que existam protocolos de comunicação específicos, que viabilizem a comunicação de uma grande quantidade de dispositivos. Neste cenário, este trabalho tem por objetivo descrever o protocolo MQTT como solução para a comunicação entre clientes de IoT, e como seria a comunicação de um cliente que envia valores da temperatura ambiente e da umidade para um Cloud broker (entidade que gerencia o uso, desempenho e entrega de serviços na nuvem).

1. Introdução

A comunicação é algo que existe desde os primórdios da humanidade. Mesmo quando ainda não existia a fala e a escrita, o homem sentia a necessidade de se comunicar para transmitir ideias, caçar, viajar, reunir membros da tribo etc. Eram utilizados desenhos nas paredes das cavernas, gestos e grunhidos para que houvesse a comunicação. Com o passar do tempo, as técnicas de comunicação foram evoluindo e padrões foram sendo desenvolvidos e disseminados, facilitando que membros de locais ou tribos diferentes pudessem se comunicar.

Em 1969, a Arpanet (*Advanced Research Projects Agency*) (Comer, 2015) revolucionou a forma de comunicação humana. Ela foi criada devido à necessidade do governo dos Estados Unidos da América estabelecer comunicação através de computadores em diferentes lugares, evitando perda de informações e de interoperabilidade caso parte da infraestrutura de comunicação sofresse um ataque. Passada a guerra fria, a Internet deixou de ser uma ferramenta militar e passou a ser um produto. As pessoas começaram a ter acesso à rede e o crescimento de conexões foi exponencial.

Além do grande número de pessoas que se conectam a rede mundial de computadores, no cenário atual, existe uma grande quantidade de dispositivos conectados. A conexão de aparelhos inteligentes à Internet caracteriza a chamada Internet das Coisas - IoT, isto é, dispositivos físicos equipados com componentes eletrônicos, sensores e atuadores com conectividade de rede que permitem aos mesmos coletarem e trocarem dados, cooperando com pessoas e o ambiente (Atzori, 2010).

A possibilidade de conectar objetos diversos à Internet permite que sejam projetadas uma infinidade de novas aplicações. Em ambientes residenciais, por exemplo, poderiam existir aplicações de monitoramento, com sensores de movimento, ou associadas ao uso de aparelhos eletrodomésticos ou até mesmo às lâmpadas. Ainda nesse ambiente, considerando a possibilidade de enviar informações aos objetos, pode ser possível controlá-los de qualquer lugar do planeta, abrindo portas, acendendo lâmpadas, ligando autofalantes, abrindo cortinas e aumentado ou diminuindo a temperatura, por exemplo. Outros ambientes também podem se beneficiar dessa tecnologia, como as industrias, comércio, acadêmico e muitos outros.

No entanto, para que as aplicações possam ser executadas nesse ambiente, é necessário uma infraestrutura complexa, com diversas tecnologias trabalhando em conjunto a fim de suportar as aplicações. Em particular, este trabalho foca na análise dos protocolos de comunicação entre os dispositivos. Em particular, é analisado o protocolo MQTT (*Message Queue Telemetry Transport*) (OASIS, 2014), cujas funcionalidades são apresentadas na seção a seguir. Posteriormente, a fim de analisar em maiores detalhes esse protocolo, a terceira seção apresenta a implementação de uma aplicação para testes e, finalmente, a quarta seção descreve as conclusões e os trabalhos futuros.

2. Protocolos para Comunicação IoT

2.1 Protocolo HTTP – *HyperTextTransferProtocol*

No próximo ano a World Wide Web, também conhecida como WWW ou simplesmente Web, comemora o seu trigésimo aniversário. Até hoje a Web é reconhecida como a principal aplicação da Internet, afinal, entre 1991 e 1995 as aplicações Web tornaram-se a maior fonte de tráfego na Internet (Comer, 2015). Graças a Web, a Internet evoluiu de um mecanismo para interligação de informações militares e acadêmicas para aplicações de vídeo sobre demanda (VoD), redes sociais e comércio eletrônico. Na verdade, atualmente, são incontáveis os serviços que podem ser acessados na Web.

A Internet, desde a adoção da arquitetura TCP/IP (Comer, 2015), não preconiza um protocolo específico para a camada de aplicação (Comer, 2015). Ao contrário, sendo o objetivo desta camada fornecer serviços específicos para as aplicações, infinitos protocolos podem ser desenvolvidos a fim de atender aos requisitos de diferentes aplicações. Alguns desses protocolos, no entanto, são empregados em aplicações tão importantes que são padronizados pelo IETF[1] (*Internet Engineering Task Force*). Esse é justamente o caso do HTTP (Hypertext Transfer Protocol) (IETF, 2014) e, posteriormente, do HTTPS (Hypertext Transfer Protocol Secure) (IETF, 2000), ambos empregados em aplicações na Web.

O HTTP é um protocolo do tipo requisição e resposta sem estado, isto é, cada requisição é interpretada independentemente das requisições anteriores ou mesmo posteriores. Sendo do tipo requisição e resposta, a comunicação, normalmente, é iniciada a partir do cliente, que muitas vezes é desconhecido ao servidor. O cliente, então, uma vez realizada a requisição, aguarda a resposta do servidor. Para cada cliente, o servidor realiza o atendimento individual da requisição que é suportada por uma conexão do protocolo TCP (Transmission Control Protocol) (IETF, 1981b), implementado na camada imediatamente inferior, a camada de transporte (Comer, 2015).

As características do protocolo HTTP, embora apropriadas para a Web, não atendem plenamente aos requisitos da comunicação IoT, uma vez que são aplicações distintas. No ambiente IoT, o número de dispositivos pode ser numeroso e as falhas de comunicação comuns e com alta latência. Nesse cenário, comunicações assíncronas, onde não se espera o retorno das mensagens, podem ser mais eficientes. Além disso, deve ser possível enviar informações aos dispositivos sem requisições prévias, o que

[1] www.ietf.org

não acontece nas versões 1.0 e 1.1 do HTTP (IETF, 2014), que são amplamente empregadas na Web. Por fim, o HTTP é tipicamente um protocolo de comunicação *unicast*, onde um cliente comunica-se com apenas um servidor por vez. Já no ambiente IoT pode ser comum enviar e receber mensagens a todos os dispositivos, caracterizando um ambiente *broadcast*, ou até mesmo *multicast*, caso a mensagem tenha por objetivo atingir um conjunto específico de dispositivos.

Adicionalmente às características já mencionadas, o cabeçalho do HTTP é considerado repetitivo e sobrecarregado (IETF, 2015), gerando um tráfego desnecessário que, no ambiente IoT pode não ser suportado. Para tentar melhorar essa característica, uma segunda versão do HTTP (HTTP/2) foi proposta (IETF, 2015). Nessa nova versão, a utilização do protocolo TCP na camada de transporte é otimizada, diminuindo a latência percebida pelos clientes. Além disso, o conteúdo é transportado em um formato binário, o que geralmente diminui a quantidade de dados transmitidos em razão da compactação. Finalmente, é interessante destacar que o HTTP/2 permite o envio de dados do servidor independente das solicitações dos clientes (PUSH).

Assim, o HTTP/2, quando comparado ao HTTP, é teoricamente mais apropriado para o ambiente IoT, uma vez que as suas transmissões são compactas e com mais baixo *overhead* o que, certamente, é mais apropriado em um ambiente onde os dispositivos podem ser limitados em termos de hardware e consumo de energia. No entanto, para essa comunicação existe um outro protocolo específico, o MQTT, que será apresentado na próxima seção.

2.2 Protocolo MQTT – *Message Queue Telemetry Transport*

O protocolo MQTT foi desenvolvido pela IBM[2] mas, atualmente, é um padrão aberto para comunicação entre dispositivos (OASIS, 2014). Diferente de outros protocolos como, por exemplo, o HTTP anteriormente mencionado, a arquitetura do MQTT é do tipo publicação e assinatura (*publish-subscribers*). Nessa arquitetura, o dispositivo é responsável por enviar (*publish*) as informações ao servidor, que opera como um intermediário (*broker*). Tendo conhecimento dos clientes que estão interessados nas informações enviadas (*subscribers*), o *broker* retransmite as informações recebidas.

O MQTT é um protocolo situado na camada de aplicação da arquitetura TCP/IP, conforme apresentado na Figura 2. Ele define o modelo de operação entre os equipamentos, especificando os papéis de cada um, o formato das mensagens e a ordem entre elas. Adicionalmente, as funcionalidades da rede são providas pelas camadas imediatamente anteriores, com destaque para o protocolo TCP, para o protocolo IP (IETF, 1981a) e para as diferentes tecnologias de comunicação que estão na camada inferior dessa arquitetura (ethernet, wifi etc.).

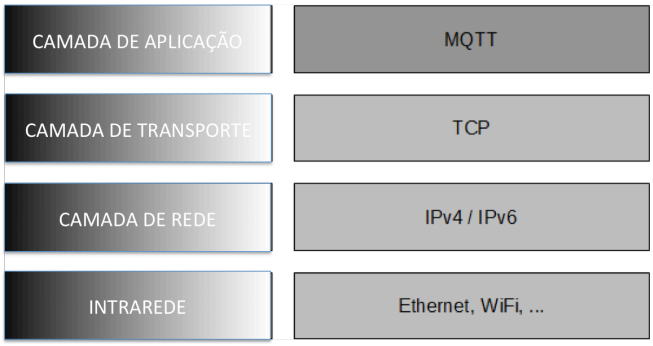

Figura 2. Distribuição dos Protocolos nas Camadas (do autor)

[2] www.ibm.com

O TCP é o responsável pela entrega confiável dos dados. Isto é, quando a mensagem é enviada sua entrega é garantida ao destino. Diferente do HTTP, o MQTT também pode trabalhar sobre outro protocolo da camada de transporte, o UDP (IETF, 1980). Apesar de não garantir a entrega dos dados, o protocolo UDP exige um menor controle da transmissão e, com isso, pode ser mais adequado para dispositivos com menores recursos ou ainda nas aplicações onde o atraso embutido pela retransmissão dos dados não é tolerado.

Através do protocolo IP os dados transmitidos podem ser enviados em toda a Internet. Assim, da mesma forma que o HTTP, o MQTT pode ser utilizado com dispositivos ou *brokers* em qualquer lugar onde a Internet esteja disponível. Particularmente, o *broker* pode ser oferecido aos usuários como um serviço hospedado em um servidor remoto na Internet. Do ponto de vista da infraestrutura de comunicação, o MQTT pode ser implementado sobre qualquer tecnologia. Assim, podem existir dispositivos que se comunicam através de cabos de rede, sobre redes sem fio ou ainda usando comunicação celular ou *bluetooth*, por exemplo.

Para conexão, primeiro o cliente conecta-se ao *broker* enviando uma mensagem CONNECT. Essa mensagem têm os parâmetros de conteúdo especificados na Tabela 1:

Parâmetro	Descrição
cleanSession	Esta sinalização especifica se a conexão é persistente ou não. Uma sessão persistente armazena todas as assinaturas e as mensagens possivelmente perdidas (dependendo do QoS) no broker. (Consulte Tabela 3 para obter uma descrição do QoS).
username	As credenciais de autenticação e autorização do broker.
password	As credenciais de autenticação e autorização do broker.
lastWillTopic	Quando a conexão for encerrada inesperadamente, o broker publicará automaticamente uma mensagem de "último desejo" em um tópico.
lastWillQos	O QoS da mensagem de "último desejo". (Consulte Tabela 3 para obter uma descrição do QoS).
lastWillMessage	A própria mensagem de "último desejo".
keepAlive	Este é o intervalo de tempo em que o cliente precisa efetuar ping no broker para manter a conexão ativa.

Tabela 1. Parâmetros da mensagem CONNECT

Em resposta a mensagem CONNECT, o cliente recebe uma mensagem CONNACK, que possui os parâmetros listados na Tabela 2.

Parâmetro	Descrição
sessionPresent	Indica se a conexão já tem uma sessão persistente. Ou seja, a conexão já tem tópicos assinados e receberá a entrega de mensagens ausentes.
returnCode	0 indica sucesso. Outros valores identificam a causa da falha.

Tabela 2. Parâmetros da mensagem CONNACK

Assim que a conexão é estabelecida, o cliente pode enviar uma ou mais mensagens SUBSCRIBE ao *broker*, para indicar sobre quais tópicos ele deseja receber informações. Cada tópico corresponde a um item, como a temperatura, umidade, entre outros. A Tabela 3 contém os parâmetros da mensagem SUBSCRIBE.

Parâmetro	Descrição
QoS	Indica com que consistência as mensagens neste tópico precisam ser entregues aos clientes. Valor 0: não confiável, a mensagem é entregue no máximo uma única vez, se o cliente estiver indisponível no momento, ele perderá a mensagem. Valor 1: a mensagem deve ser entregue pelo menos uma vez. Valor 2: a mensagem deve ser entregue exatamente uma vez.
tópico	Um tópico para assinar. Um tópico pode ter vários níveis separados pelo caractere barra. Por exemplo, "dw/demo" e "ces/engenhariasoftware/mqtt" são tópicos válidos.

Tabela 3. Parâmetros da mensagem SUBSCRIBE

Em resposta a assinatura de um tópico, o broker retornará uma mensagem SUBACK, com o parâmetro returnCode, de acordo com os valores listados na Tabela 4.

Parâmetro	Descrição
returnCode	Existe um código de retorno para cada um dos tópicos no comando SUBSCRIBE. Os valores de retorno são os seguintes: Valores 0 a 2: sucesso como nível de QoS correspondente. (correspondente aos valores listados na Tabela 3). Valor 128: falha

Tabela 4. Parâmetros da mensagem SUBACK

Cada tópico que o cliente solicita receber os valores pode ser posteriormente removido, através de uma mensagem UNSUBSCRIBE, que pode ser relacionada a um ou vários tópicos, especificados na mensagem.

Além de receber mensagens, os clientes também podem enviar mensagens ao *broker*. Essa operação no MQTT é realizada através da mensagem PUBLISH. Cada mensagem contém um tópico e um valor, conforme a Tabela 5. Quando recebe uma mensagem PUBLISH, o *broker* encaminha os dados dessa mensagem a todos os clientes que assinam esse tópico.

Parâmetro	Descrição
topicName	O tópico no qual a mensagem é publicada.
QoS	O nível de qualidade de serviço da entrega da mensagem. (de acordo com os valores da Tabela 3).
retainFlag	Esta sinalização indica se o *broker* reterá a mensagem como a última mensagem conhecida deste tópico.
“carga útil”	Os dados reais na mensagem. Pode ser uma sequência de texto ou valores binário de dados.

Tabela 5. Parâmetros da mensagem PUBLISH

3. Aplicação

Atualmente, empresas têm investido muito em segurança preventiva, com o objetivo de evitar que acidentes aconteçam e venham prejudicar o seu negócio. Assim, é possível encontrar no mercado diversos equipamentos e sensores de automação que servem para monitorar determinado ambiente e enviar as informações necessárias para a tomada de decisões.

Independente do setor econômico como, por exemplo, saúde, alimentação, segurança, comunicação, entre outros, é possível encontrar um ambiente que precisa ser controlado. Em ambientes dependentes da tecnologia, por exemplo, usualmente um ambiente a ser controlado é o *Data Center*, um ambiente onde diversos equipamentos de tecnologia estão disponíveis e onde os dados são armazenados e processados. Esses ambientes precisam ser constantemente monitorados, uma vez que precisam operar em condições ideias.

Na verdade, o monitoramento, é a base da atividade de gerenciamento, que consistem em controlar as atividades importantes. No gerenciamento, os dados são constantemente coletados, oferecendo ao gerente as informações monitoradas, criando uma base de conhecimento para que decisões posam ser corretamente tomadas, a partir dos indicadores apropriados.

Alguns fatores de segurança de um data center, por exemplo, que podem ser importantes para o monitoramento, são a temperatura e a umidade do ar (Zucchi, 2013). Segundo a ASHRAE[3] (*American Society of Heating, Refigerating and Air Conditioning Engineers*) é ideal que a temperatura de data centers esteja entre 18º C e 27° C e que a umidade relativa do ar esteja entre 40% e 55%. Para realizar esse controle, uma aplicação IoT pode ser desenvolvida, utilizando o protocolo MQTT para realizar a comunicação entre os objetos que capturam a temperatura e a umidade do ar, o *broker*, que concentra essas informações, e os objetos responsáveis pelas operações corretivas, caso os valores ideais não sejam alcançados.

A Figura 3 ilustra o funcionamento da aplicação proposta e os papeis dos principais componentes para a realização dessa aplicação. Um objeto que contém a informação da temperatura publica a informação no *broker* através do protocolo MQTT. O *broker*, por sua vez, publica essa informação em todos os clientes que subscreveram o tópico temperatura (temp).

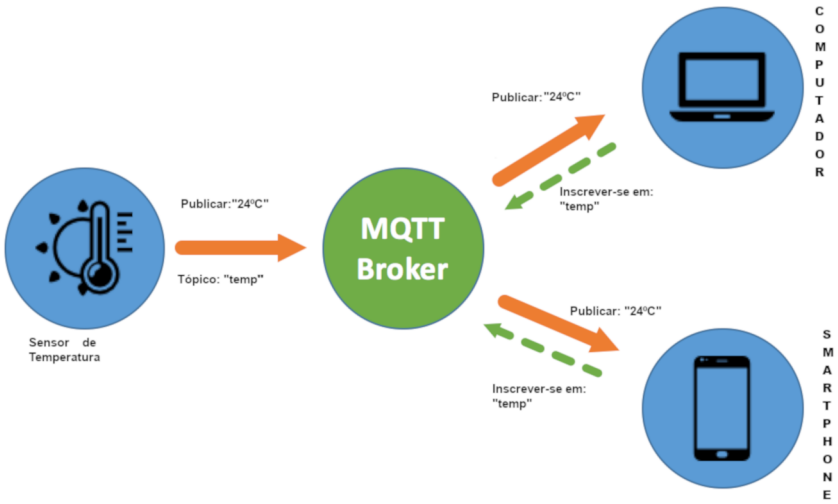

Figura 3. Funcionamento do Broker na Aplicação (fonte: www.ibm.com)

3.1 Broker

O *broker* é um elemento fundamental na arquitetura *publish-subscribers*. Dependendo da aplicação, pode ser necessário que esse elemento manipule milhões de mensagens simultâneas, com origem e destino em diversos clientes. O *broker* é responsável por receber todas as mensagens, filtrá-las, determinar quem deve receber cada mensagem e enviar essas mensagens para os destinos.

[3] www.ashrae.org

Outra responsabilidade do *broker* é autenticar e autorizar os clientes a enviar e/ou receber as mensagens. Assim, o *broker* pode criar e manter sessões dos clientes autenticados, evitando que, a cada nova mensagem, a autenticação necessite ser realizada. Mensagens de conexão (início ou término) e de sessão são oferecidas pelo protocolo MQTT e foram apresentadas na Seção 2.

O Eclipse Mosquitto[4] é um exemplo de um *broker*, implementado com código fonte aberto e que oferece suporte ao protocolo MQTT nas versões 3.1 e 3.1.1 (OASIS, 2015). Ele pode ser obtido gratuitamente (mosquito.org) e instalado nas mais diversas organizações. Adicionalmente, é possível encontrar diversos serviços na Internet que oferecem *brokers* online que utilizam a implementação Eclipse Mosquitto. Como exemplo, podem ser citados o Amazon IoT (aws.amazona.com/iot), o Microsoft Azure (azure.microsoft.com), o HiveMQ (www.hivemq.com), o CloudMQTT (cloudmqtt.com), entre outros.

A utilização de um *broker* na Internet (na nuvem), além de oferecer um serviço sem a necessidade da existência de um servidor dedicado a esse fim, permite que os clientes possam acessar esse serviço facilmente, a partir de qualquer ponto da Internet. A seguir é apresentado um desses *brokers*, o CloudMQTT.

3.1.1 CloudMQTT

O CloudMQTT é uma implementação na nuvem do *broker* Mosquito que atualmente utiliza servidores da Amazon[5] como suporte a essa aplicação. Depois do registro do usuário no sistema, é possível criar diversas instâncias, onde cada uma corresponde a um *broker*. A Figura 4 apresenta a tela de criação da instância *Controller*, utilizada como exemplo neste trabalho. Nesse momento, além do nome da instância, é necessário escolher o plano de contratação que, neste caso, foi o plano *CuteCat*, que é gratuito e atende os objetivos demonstrativos deste trabalho.

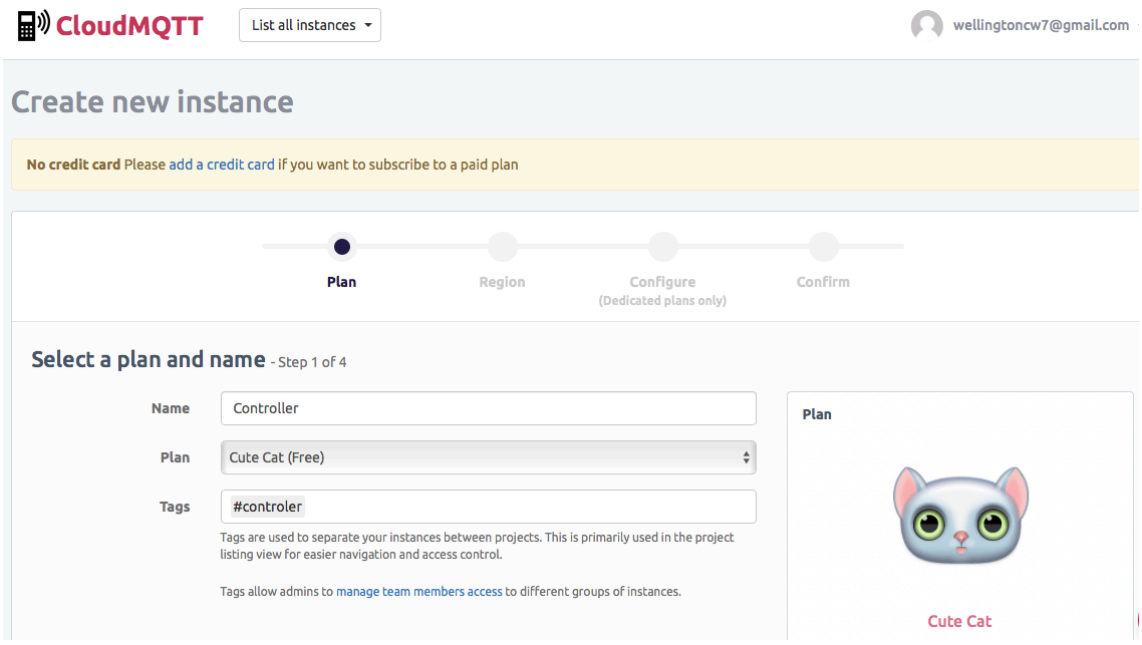

Figura 4. Criação do Broker on-line no CloudMQTT (do autor)

Na criação do *broker* no CloudMQTT é necessário especificar a região do servidor que irá prover o *broker*. Para o plano gratuito, na ocasião de realização deste

[4] mosquito.org
[5] aws.amazon.com

trabalho, apenas os servidores da Irlanda e do norte da Virgínia nos Estados Unidos estavam disponíveis, sendo escolhido esse último, conforme Figura 5.

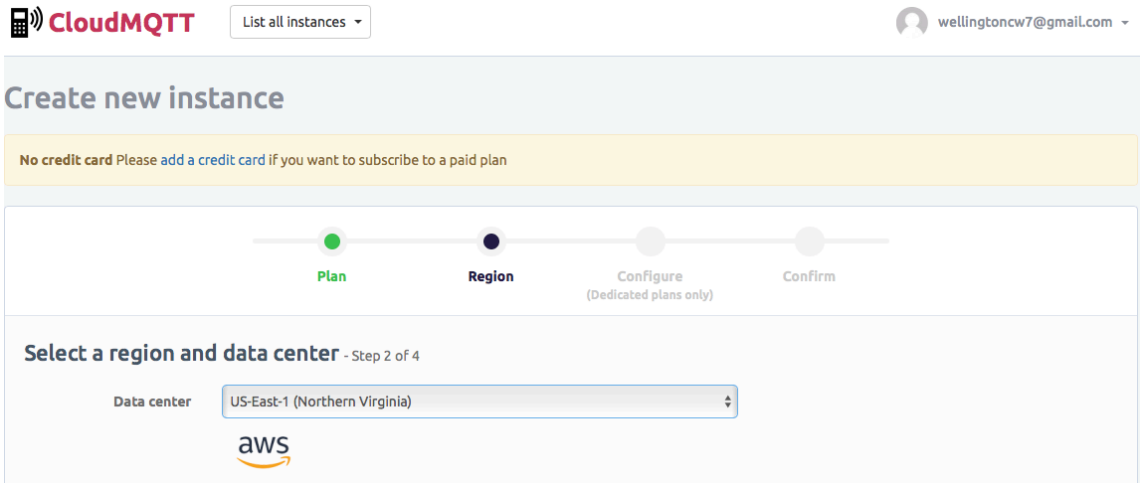

Figura 5. Escolha da Região do Servidor do Broker on-line no CloudMQTT (do autor)

Após as configurações mencionadas o *broker* está pronto e funcionando. A Figura 6 apresenta os seus detalhes. O acesso dos clientes para a postagem de valores deve ser realizado através da url m15.cloudmqtt.com. O usuário e a senha para autenticação também são apresentados na Figura 6. O acesso através do protocolo MQTT é realizado através da porta 11529. O acesso também pode ser realizado de forma segura, utilizando o protocolo SSL (*Secure Sockets Layer*) (IETF, 2011a) sob o MQTT, através da porta 21529, de tal forma que os dados transmitidos são criptografados. Finalmente, é possível também realizar o acesso a esse *broker* utilizando *WebSockets* (IETF, 2011b), através da porta 31529. Nesse caso, os dados são enviados usando o HTTP e criptografados utilizando o protocolo TLS (*Transport Layer Security*) (IETF, 2018).

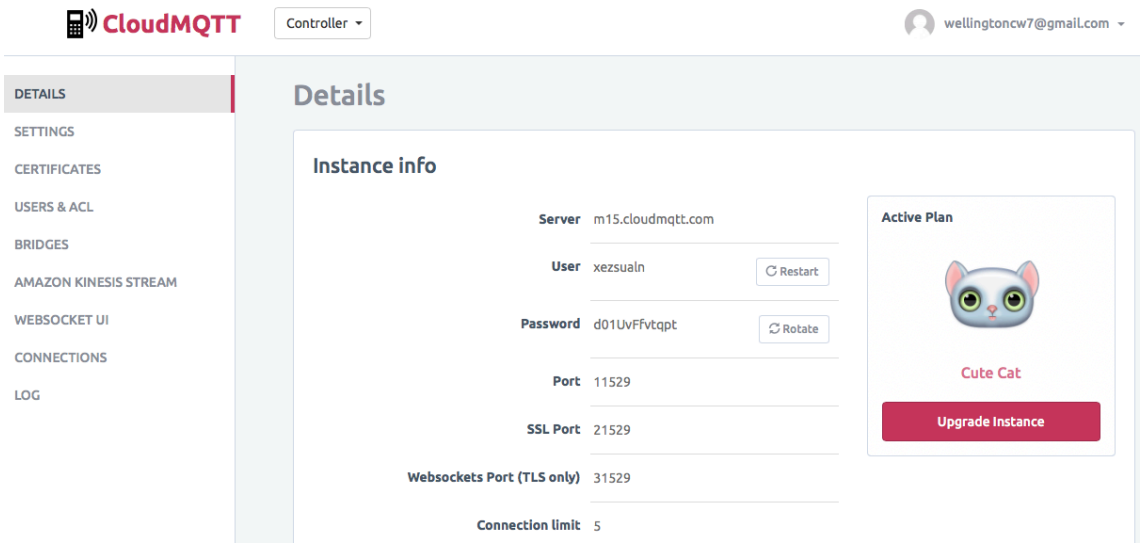

Figura 6. Parâmetros do BrokerController, disponível on-line no CloudMQTT (do autor)

O CloudMQTT permite outras configurações como a especificação de outros usuários, regras de acesso específicas para cada usuário e utilização de certificados digitais para validação de autenticação. É possível acompanhar, em tempo real, os clientes conectados a esse *broker*. Também está disponível o registro de todas as operações realizadas.

3.2 Clientes

Na arquitetura *publish-subscriber* para IoT, apesar da importância do *broker*, o cliente é a parte fundamental. Nessa arquitetura, os dados têm origem e destino nos clientes. A seguir, são apresentados dois clientes, um já pronto, o IoT MQTT Dashboard[6], e um cliente desenvolvido neste trabalho.

3.2.1 IoT MQTT Dashboard

O IoT MQTT *Dashboard* é um aplicativo mobile para o sistema operacional Android[7]que pode ser utilizado para demonstração da comunicação em um cenário de IoT. Entre as suas funcionalidades, destacam-se a possibilidade de utilização de SSL e utilização de gráficos. Também é possível utilizar componentes como botões, áreas de texto e barras para envio de informações ao *broker*. Na tela inicial do IoT MQTT *Dashboard* é possível estabelecer a conexão ao *broker* através do "botão flutuante", com o símbolo de "mais", apresentado na Figura 7.

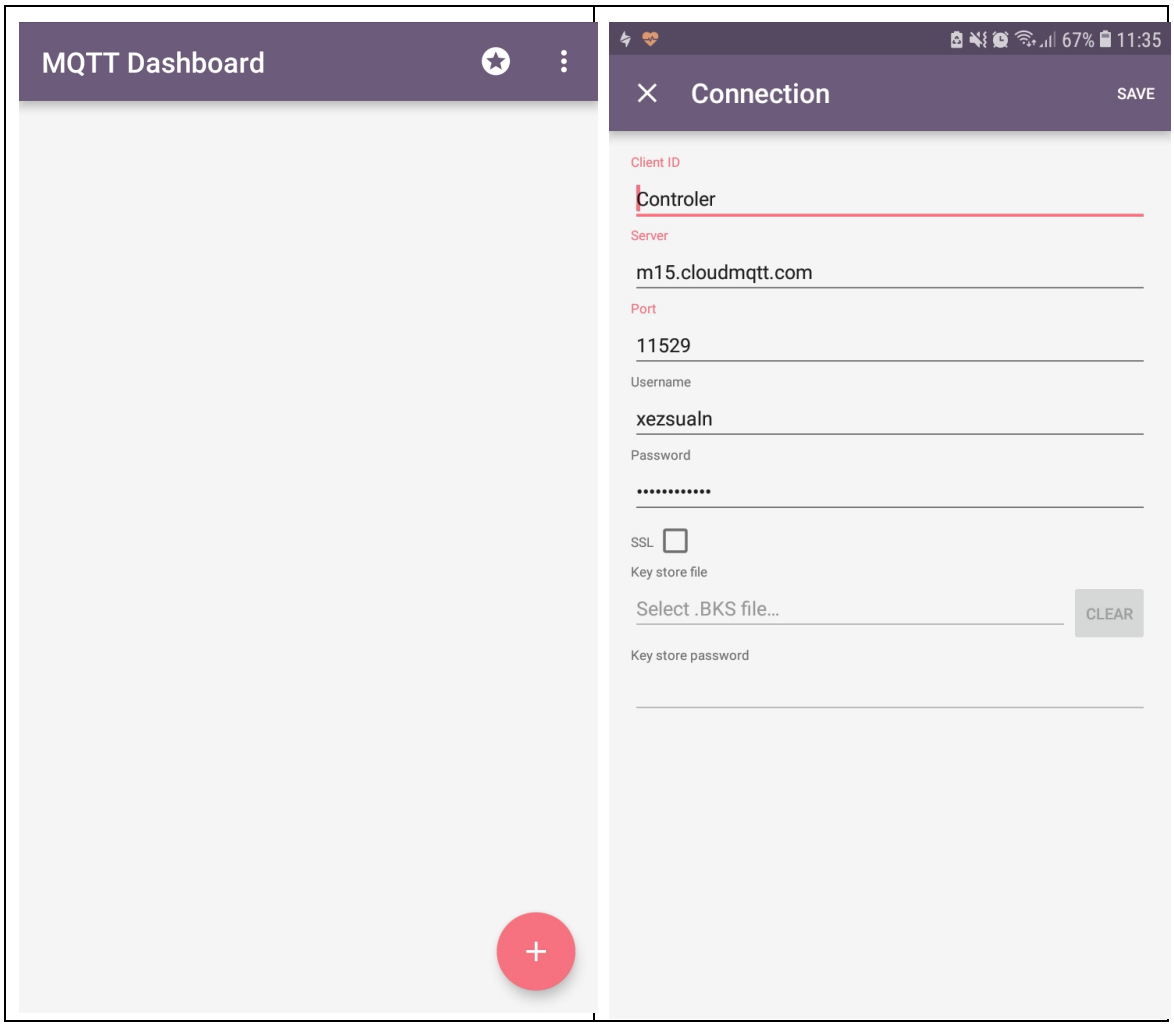

Figura 7. Parâmetros do Cliente, no Aplicativo IoT MQTT DashBoard (do autor)

[6] https://play.google.com/store/apps/details?id=com.thn.iotmqttdashboard&hl=en
[7] https://www.android.com

Também na Figura 7, após a solicitação da criação de uma nova conexão, é apresentada a tela para especificação dos parâmetros de conexão. Esses parâmetros são aqueles especificados pelo *broker*, apresentados anteriormente na Figura 6.

Especificada a conexão, o IoT MQTT *Dashboard* oferece as opções de *Publish* e *Subscribe*, onde é possível, respectivamente, enviar mensagens ao *broker* e recebe-las. Considerando, por exemplo, um ambiente de *Data Center*, onde a temperatura ideal varia entre 18º C e 27° C, com umidade relativa do ar entre 40 e 55%, foram criados os indicadores "**temp**" (temperatura) e "**umd**" (umidade) para exemplificar a publicação e recebimento das mensagens utilizando o protocolo MQTT.

A Figura 8 apresenta a postagem de possíveis valores referentes a temperatura e a umidade. Dentro do aplicativo foram criados botões com valores de temperatura e uma barra de rolagem que indica o valor da umidade. Quando os valores são escolhidos, e o botão "**PUBLISH**" é selecionado, os valores são enviados ao *broker* através do MQTT. O *broker*, ao receber esses valores, repassa os mesmos para todos que solicitaram serem notificados por modificações nesses tópicos. Nesse caso, o próprio aplicativo também recebe as notificações, conforme apresentado na Figura 8.

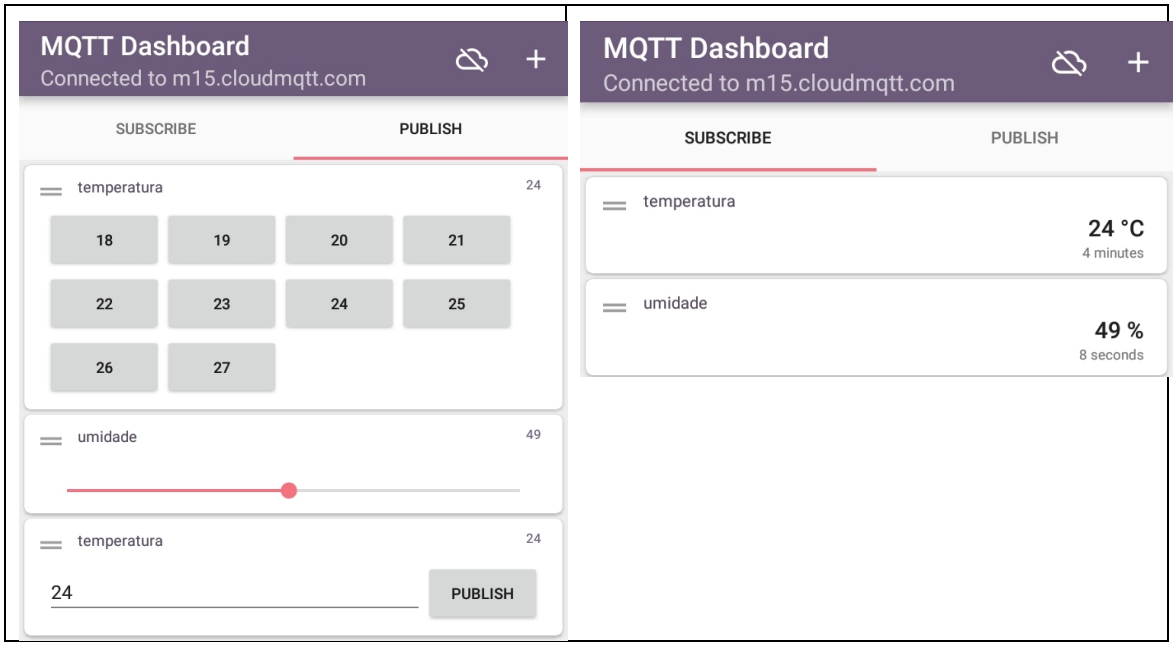

Figura 8. Publicação e Recebimento de Valores no IoT MQTT DashBoard (do autor)

3.2.2 Aplicação IoT MQTT

O IoT MQTT *Dashboard* é um aplicativo que permite realizar testes de envio e recebimento de valores de um *broker*. Os valores postados, no entanto, não representam valores reais, obtidos a partir, por exemplo, de medidas da temperatura e da umidade. Para realizar o envio de valores reais é necessário, antes do envio propriamente dito, a coleta dessas informações.

Atualmente, a alternativa usual, muito utilizada por desenvolvedores para implementar sensores consiste na utilização do microcontrolador ESP8266 (ESPRESSIF, 2018). A esse microcontrolador, que pode ter seu funcionamento definido através de um código embarcado, inúmeros sensores podem ser acoplados, incluindo, neste caso, os sensores de umidade e temperatura. No entanto, apesar da facilidade atual de aquisição desse microcontrolador e dos seus sensores e da ampla documentação encontrada para a sua configuração, pode ser necessário, para a sua utilização, a implementação de circuitos eletrônicos que fogem do escopo deste trabalho. O sensor

de umidade, por exemplo, envia um sinal analógico, que precisa de um circuito eletrônico para converter o sinal ao microcontrolador.

A alternativa escolhida neste trabalho consiste em utilizar os próprios sensores disponíveis no celular para publicar os valores no broker. No aparelhos Android, a classe Sensor é utilizada para especificar todos os sensores disponíveis no aparelho. Como cada aparelho pode oferecer um conjunto diferente de sensores, a classe SensorManager pode receber todos os sensores disponíveis no aparelho. A Figura 9 apresenta um trecho de código onde, na linha 14 o SensorManager é instanciado e, na linha subsequente, uma lista dos sensores disponíveis é coletada e colocada em uma lista (linhas 17 a 20)

```
...
14.   sensorManager= (SensorManager) getSystemService(SENSOR_SERVICE);
15.   sensorList= sensorManager.getSensorList(Sensor.TYPE_ALL);
16.
17.   List<String> nomes = new ArrayList<>();
18.   for (Sensor s : sensorList) {
19.nomes.add(s.getName() + " - " + s.getVendor() + " - " + s.getType());
20.}
...
```

Figura 9. Obtenção dos sensores disponíveis no aparelho celular (do autor)

Para efetuar a conexão ao *broker* em uma aplicação Android, o projeto Eclipse Paho[8] pode ser utilizado. Esse projeto oferece duas APIs a *MqttAsyncClient* e a *MqttClient*. A primeira oferece um acesso assíncrono, de tal forma que as atividades são notificadas através de *callbacks*. A segunda, complementarmente, permite o acesso aos recursos de forma síncrona, isto é, através de chamadas realizadas pela aplicação.

A Figura 10 apresenta uma classe, chamada *MqttHelper*, que implementa a API *MqttClient* para conectar no *broker* e receber mensagens. Os parâmetros são especificados de acordo com as definições do *broker*, incluindo o endereço, a porta, o tópico e as credenciais de acesso (usuário e senha)

```
importorg.eclipse.paho.android.service.MqttAndroidClient;
importorg.eclipse.paho.client.mqttv3.DisconnectedBufferOptions;
importorg.eclipse.paho.client.mqttv3.IMqttActionListener;
importorg.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
importorg.eclipse.paho.client.mqttv3.IMqttToken;
importorg.eclipse.paho.client.mqttv3.MqttCallbackExtended;
importorg.eclipse.paho.client.mqttv3.MqttConnectOptions;
importorg.eclipse.paho.client.mqttv3.MqttException;
importorg.eclipse.paho.client.mqttv3.MqttMessage;

publicclassMqttHelper {
publicMqttAndroidClientmqttAndroidClient;

final StringserverUri= "tcp://m15.cloudmqtt.com:11529";

final StringclientId= "Controller";
final StringsubscriptionTopic= "temperatura";

final Stringusername= "xezsualn";
final Stringpassword= "d01UvFfvtqpt";

publicMqttHelper(Contextcontext){
mqttAndroidClient= new MqttAndroidClient(context, serverUri, clientId);
mqttAndroidClient.setCallback(new MqttCallbackExtended() {
@Override
publicvoidconnectComplete(booleanb, String s) {}

@Override
publicvoidconnectionLost(Throwablethrowable) {}
```

[8] www.eclipse.org/paho/clients/java

```
@Override
publicvoidmessageArrived(Stringtopic, MqttMessagemqttMessage) throwsException {
Log.w("Mqtt", mqttMessage.toString());
            }

@Override
publicvoiddeliveryComplete(IMqttDeliveryTokeniMqttDeliveryToken) {

            }
        });
connect();
    }
```

Figura 10. Classe MqttHelper, que recebe informações do broker através da API Paho (do autor)

A implementação do método connect() da classe *MqttHelper* é apresentada na Figura 11. Os parâmetros são passados para o estabelecimento da conexão.

```
...

privatevoidconnect(){
MqttConnectOptionsmqttConnectOptions = new MqttConnectOptions();
mqttConnectOptions.setAutomaticReconnect(true);
mqttConnectOptions.setCleanSession(false);
mqttConnectOptions.setUserName(username);
mqttConnectOptions.setPassword(password.toCharArray());

try{

mqttAndroidClient.connect(mqttConnectOptions, null, new IMqttActionListener() {
@Override
publicvoidonSuccess(IMqttTokenasyncActionToken) {

DisconnectedBufferOptionsdisconnectedBufferOptions = new DisconnectedBufferOptions();
disconnectedBufferOptions.setBufferEnabled(true);
disconnectedBufferOptions.setBufferSize(100);
disconnectedBufferOptions.setPersistBuffer(false);
disconnectedBufferOptions.setDeleteOldestMessages(false);
mqttAndroidClient.setBufferOpts(disconnectedBufferOptions);
subscribeToTopic();
            }

@Override
publicvoidonFailure(IMqttTokenasyncActionToken, Throwableexception) {
Log.w("Mqtt", "Failedtoconnectto: " + serverUri+ exception.toString());
            }
        });


    } catch (MqttExceptionex){
ex.printStackTrace();
    }
}

...
```

Figura 11. Implementação do método connect da Classe MqttHelper (do autor)

Finalmente, para receber as mensagens do *broker*, o método *subscribeToTopic*() da classe *MqttHelper*, que é apresentado na Figura 12, é empregado. Futuramente, esse método pode ser sobrescrito, para especificar parâmetros referentes à Qualidade do Serviço e para especificar informações sobre o tópico da subscrição.

```
...

privatevoidsubscribeToTopic() {
try{
mqttAndroidClient.subscribe(subscriptionTopic, 0, null, new IMqttActionListener() {
@Override
publicvoidonSuccess(IMqttTokenasyncActionToken) {
Log.w("Mqtt","Subscribed!");
            }

@Override
publicvoidonFailure(IMqttTokenasyncActionToken, Throwableexception) {
Log.w("Mqtt", "Subscribedfail!");
            }
        });

    } catch (MqttExceptionex) {
System.err.println("Exceptionwhilstsubscribing");
ex.printStackTrace();
    }
}

...
```

Figura 12. Implementação do método subscribeToTopic da Classe MqttHelper (do autor)

4. Conclusões e Trabalhos Futuros

Existe uma infinidade de aplicações para utilização da Internet das Coisas, que já estão revolucionando o mercado, o setor acadêmico e a indústria. Com o desenvolvimento dessas aplicações e com o advento da melhoria das tecnologias necessárias, é possível usufruir de um cenário com dispositivos inteligentes espalhados por todo o globo capazes de enviar e receber informações.

Com a utilização do protocolo MQTT para a comunicação entre clientes e servidores é possível imaginar um cenário onde um ambiente de Data Center é controlado através de um cliente que capturaria as informações de temperatura de umidade do ar e enviaria a um *broker* que repassaria as informações aos clientes assinados ao tópico de temperatura. Porém, devido a limitação de aparelhos Android, onde nem todos possuem sensores para captura de temperatura e umidade, não foi totalmente possível demonstrar a aplicação coletando a informação e enviando ao *broker*. Mesmo assim, foi possível verificar que é possível realizar essa tarefa, como demonstrado nos testes realizados com o aplicativo MQTT *Dashboard*.

Como um trabalho futuro, para continuidade desse assunto e para realizar a implementação completa do cenário proposto, poderia ser utilizado, como mencionado anteriormente, o microcontrolador ESP8266 onde inúmeros sensores podem ser acoplados, incluindo, neste caso, os sensores de umidade e temperatura, fazendo com que o cenário especificado neste trabalho esteja devidamente implementado.

Essa implementação pode até mesmo ser estendida, fazendo com que além de coletar os dados e enviar ao *broker* que irá repassar as informações aos clientes assinados no tópico, medidas de prevenção e controle possam ser desenvolvidas e serem acionadas automaticamente, caso os valores desejados, no caso desse cenário, a temperatura entre 18º C e 27° C, com umidade relativa do ar entre 40 e 55%, não sejam alcançados.

5. Referências

ATZORI, l., IERA, A., MORABITO, G. **The Internet ofThings: A Survey**. Computer Networks, 54, v. 15, pp. 2787-2805, 2010.

COMER, D. E. Interligação de redes com TCP/IP. ed. 6, 486 pp, Elsevier, Rio de Janeiro, 2015.

ESPRESSIFSystens. **SmartConnectivity Platform: ESP8266**. Disponível em: https://cdn-shop.adafruit.com/datasheets/ESP8266_Specifications_English.pdf. Acesso em: Novembro 2018.

IETF – Internet EngineeringTask Force. **UserDatagramProtocol**. Request for Comments 768. Agosto 1980.

IETF – Internet EngineeringTask Force. **Internet ProgramProtocolSpecification**. Request for Comments 791. Setembro 1981a.

IETF – Internet EngineeringTask Force. **TransmissionControlProtocol**. Request for Comments 2817. Setembro 1981b.

IETF – Internet EngineeringTask Force. **Upgradingto TLS Within HTTP/1.1**. Request for Comments 2817. Maio 2000.

IETF – Internet EngineeringTask Force. **The SSL ProtocolVersion 3.0**. Request for Comments 6101. Agosto 2011a.

IETF – Internet EngineeringTask Force. **The WebSocketProtocol**. Request for Comments 6455. Dezembro 2011b.

IETF – Internet EngineeringTask Force. **Hypertext TransferProtocol HTTP/1.1: MessageSyntaxandRouting**. Request For Comments 7230. Junho 2014.

IETF – Internet EngineeringTask Force. **Hypertext TransferProtocolVersion 2 (HTTP/2).**Request For Comments 7540. Maio 2015.

IETF – Internet EngineeringTask Force. **The TransportLayer Security (TLS) ProtocolVersion 1.3.**Request For Comments 8446. Agosto 2018.

OASIS – OASIS Standard. **MQTT Version 3.1.1.** Outubro 2014.

ZUCCHI, W.L., AMÂNCIO, A. Construindo um Data Center. Revista USP n.97, pp. 43-58. 2013.