CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Ismael Rodrigues Martins

Estudo dos Protocolos de Comunicação MQTT e CoAP para Aplicações *Machine-to-Machine* e Internet das Coisas

Americana, SP

2014

CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Ismael Rodrigues Martins

Estudo dos Protocolos de Comunicação MQTT e CoAP para Aplicações *Machine-to-Machine* e Internet das Coisas

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso de Tecnologia em Análise de Desenvolvimento de Sistemas, sob a orientação do Prof. Dr. José Luís Zem

Área de concentração: Sistemas de Computação

Americana, SP

2014

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

Martins, Ismael Rodrigues
M343e Estudo dos protocolos de comunicação MQTT e CoAP para aplicações *machine-to-machine* e Internet das Coisas. / Ismael Rodrigues Martins. – Americana: 2014.
55f.

Monografia (Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Dr. José Luís Zem

1. Comunicação de dados I. Zem, José Luís II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.519

Ismael Rodrigues Martins

Estudo dos Protocolos de Comunicação MQTT e CoAP para Aplicações Machine-to-Machine e Internet das Coisas

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de SIstemas pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Sistemas de Computação.

Americana, 03 de dezembro de 2014.

Banca Examinadora:

José Luís Zem (Presidente)
Doutor
Fatec Americana

Wagner Siqueira Cavalcante (Membro)
Mestre
Fatec Americana

Moacir Degasperi Júnior (Membro)
Doutor
Fatec Americana

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus pela força, por ter me trazido até aqui e me dado a graça de concluir mais esta etapa em minha vida. Agradeço aos meus pais pela educação que me deram, pelo lar, e acima de tudo, uma família onde sempre pude me apoiar. Agradeço aos professores que passaram pela minha carreira acadêmica e todo apoio e incentivo que sempre me deram. Agradeço ao professor José Luís Zem pela orientação neste trabalho, por dedicar uma parcela de seu precioso tempo para me ajudar. Agradeço também aos colegas que estão comigo desde o inicio do curso, por seu companheirismo, amizade e cumplicidade, sobretudo ao colega e amigo Álvaro Augusto Roberto, sempre disposto a ouvir meus devaneios e projetos mirabolantes que nunca consigo tirar do papel. Agradeço aos colegas do Instituto de Computação da Unicamp pelo apoio, e por tudo que me ensinaram nos últimos dois anos.

DEDICATÓRIA

Aos meus pais, minha maior fonte de inspiração.

RESUMO

O presente trabalho aborda os assuntos de Internet das Coisas e aplicações *Machine-to-Machine,* especialmente no que concerne à comunicação entre dispositivos na tecnologia M2M, trazendo uma definição destes conceitos com base em trabalhos já publicados. A monografia objetiva realizar um estudo comparativo entre dois protocolos de comunicação voltados para aplicações M2M: de um lado o *Message Queue Telemetry Transport* (MQTT) desenvolvido pela IBM em parceria com a *Eurotech*, e de outro o CoAP (*Constrained Application Protocol*), que recentemente tournou-se um RFC (7252), desenvolvido pelo grupo de trabalho do IETF denominado *Constrained RESTful Environments*, ambos voltados para equipamentos pequenos e com restrições de conexão e processamento. Para isso, foi realizado um levantamento das características de cada protocolo, bem como seu funcionamento no que se refere à troca de mensagens. Para fazer a comparação foram desenvolvidas duas aplicações (uma para cada protocolo) executando em um ambiente controlado com a principal função de transmitir informações simuladas de temperatura na rede. O tráfego gerado por esta transmissão foi capturado pelo *Wireshark* e após analisar os resultados concluiu-se que apesar de ambas as aplicações funcionarem de acordo com o esperado, análises baseadas apenas na informação dos pacotes capturados pelo *Wireshark* não foram suficientes para apontar qual dos dois protocolos estudados é a melhor opção para uma aplicação M2M. Ainda assim, outras formas de análise são sugeridas na tentativa de responder a questão: um ambiente real, semelhante àquele descrito pela documentação dos protocolos e uma análise de desempenho dos protocolos considerando todo o caminho percorrido pelo dado transmitido. Por fim, o trabalho encoraja os analistas, desenvolvedores e outros profissionais da área de TI a dar seguimento aos estudos que abordem os assuntos de aplicações M2M e Internet das Coisas.

Palavras Chave: Internet das coisas, aplicação M2M, protocolo de comunicação.

ABSTRACT

The present paper addresses the subjects of Internet of Things (IoT) and Machine-to-Machine (M2M) applications, especially regarding the communication among devices in the M2M technology, bringing up a definition for these terms based on papers already published. This monograph objectifies perform a comparative study between two communication protocols for M2M applications: in one side is Message Queue Telemetry Transport (MQTT) developed by IBM along with Eurotech, and in the other side is the Constrained Application Protocol (CoAP) which has become an RFC (7252) recently and was developed by a IETF workgroup named Constrained RESTful Environments, both protocols suitable for little devices facing connection and processing constraints. To do this, a gathering of features from each one of the protocols was performed, as well as it was with the functionalities concerning message exchange. For the comparison, two applications were developed (one for each protocol) which was executed in a controlled environment with the main function of transmit a simulated temperature information in the network. The traffic generated by this transmission was captured by Wireshark and after an analysis of the results it was concluded that despite both applications worked as well as expected, the analysis based only in the information in the packages captured by Wireshark were not enough to determine which one of the two protocols studied is the best option for a M2M application. Even so, other forms of analyses are suggested in the attempt to try to answer the question: a real environment, as that described in the protocols documentation and a performance analysis of the protocols considering all the way traversed by the data in the transmission. Finally, the paper encourages the analysts, developers and others IT professionals to continue the studies concerning the subject of M2M applications and Internet of Things

Keywords: *Internet of Things, M2M application, Communication protocol.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACK: Acknowledgment.
CoAP: Constrained Application Protocol.
CoRE: Constrained RESTful Environments.
DTLS: Datagram Transport Layer Security.
ETSI: European Telecommunications Standards Institute.
FTP: File Transfer Protocol.
GPS: Global Positioning System.
HTTP: Hyper Text Transfer Protocol.
IETF: Internet Task-Force Engineering.
IoT: Internet of Things.
IP: Internet Protocol.
IPV6: Internet Protocol Versão 6.
M2M: Machine-to-Machine.
MQTT: Message Queue Telemetry Transport.
PC: Personal Computer.
PDU: Protocol Data Unit.
QoS: Quality of Service.
REST: Representational State Transfer.
RFC: Request for Comments.
RFID: Radio-Frequency Identification.
ROI: Return of Investment.
SFTP: SSH File Transfer Protocol.
TC: Technical Comitee.
TCP: Transmission Control Protocol.
TLV: Type-Length-Value.
TMN: Telecommunications Management Network.
TS: Technical Specification.
UDP: User Datagram Protocol.
URI: Universal Resource Identifier.

SUMÁRIO

1. INTRODUÇÃO

A Internet das Coisas é um paradigma que vem ganhando espaço rapidamente no cenário moderno das telecomunicações *wireless,* segundo ATZORI, IERA e MORABITO (2010). A ideia consiste em poder conectar uma variedade de objetos (sensores, *tags RFID*, *smartphones*, computadores, e até objetos de uso mais comum) entre si. Estes objetos geram um fluxo de dados e a conexão permite que eles transmitam estes dados para outros objetos no meio, formando assim uma internet de coisas. Este termo confunde-se com o termo 'Comunicações *Machine-to-Machine*'(M2M *Communications*), que trata da comunicação máquina a máquina, ou seja, entre dois ou mais dispositivos, sem um 'usuário final' envolvido no processo da troca de informações. Com esta comunicação abre-se um leque imenso de aplicações que podem, entre outras coisas, registrar, tratar e manipular os dados gerados e transmitidos pelos objetos que estão interligados. Como por exemplo, uma aplicação que recebe de maneira contínua dados de um sensor que mede a temperatura de um ambiente e, com base nos dados obtidos, pode gerar estatísticas que descrevem as leituras do sensor em um espaço de tempo para depois emitir um alerta via email ou SMS para um ou mais indivíduos caso a temperatura tenha atingido níveis muito altos, ou até publicar estas informações para outro dispositivo que poderia utilizá-las de outra forma, entre outras coisas.

As aplicações voltadas à M2M têm potencial para se tornar uma tendência no desenvolvimento de softwares nos próximos anos tendo em vista a variedade de áreas, tanto do setor da indústria, quanto de utilidade doméstica que poderão fazer uso de uma solução automatizada que possa integrar os dispositivos que compõem seu ambiente, além de também trazer à tona a realidade da Internet das Coisas. Quando se pensa no porquê de estudar a comunicação em aplicações M2M, pode-se enumerar que:

- M2M é um assunto em expansão;
- A computação está ficando cada dia mais ubíqua, ou seja, em toda parte. Essa imensa disponibilidade de conectividade é um cenário ideal para a exploração de tecnologias M2M;
- Esta ubiquidade na computação traz também um momento propício para investir em tecnologias que coloquem não apenas supercomputadores na rede, mas também dispositivos restritos, de pequeno porte, com baixo consumo de energia elétrica e baixa capacidade computacional (como memória e processamento);

- Com o advento do IPV6 (*Internet Protocol* Versão 6), a capacidade para endereçamento de nós na rede irá possibilitar um número imenso de dispositivos conectados;

As informações coletadas e fornecidas pelos dispositivos através das aplicações M2M podem ser combinadas, retransmitidas e utilizadas para que as pessoas possam entender melhor o ambiente à sua volta, começando com a própria casa. Logo, aplicações M2M tem um forte impacto na sociedade contemporânea;

Em junho de 2014 foi publicado o RFC 7252, que propõe o CoAP (*Constrained Application Protocol*) como um padrão para protocolo de troca de mensagens voltado para dispositivos com restrições computacionais e de energia bem como para redes com baixa largura de banda. O documento é de autoria de Zach Shelby, K. Hartke e C. Bormann e apesar da recente publicação do RFC, existem poucos estudos sobre ele no tocante ao uso do CoAP para aplicações M2M (IETF, 2014).

Existe outro protocolo de troca de mensagens voltado para aplicações M2M: o MQTT (*Message Queuing Telemetry Transport*) desenvolvido pela IBM em 1999. Sua proposta é atuar principalmente, mas não exclusivamente, em ambientes onde há baixa largura banda ou redes instáveis bem como em dispositivos embarcados com recursos limitados de memória e processamento (PIPER, 2013).

Colocando estes dois protocolos em uma comparação superficial, de um lado tem-se um protocolo pouco conhecido, com poucas implementações que não são muito difundidas na comunidade desenvolvedora, mas que possui um RFC como um padrão proposto; do outro lado, temos um protocolo criado já há algum tempo, com diversas implementações, conhecido e utilizado em inúmeros protótipos, mas que nunca foi padronizado.

O objetivo geral deste trabalho é fazer um estudo comparativo entre o CoAP e o MQTT para indicar qual deles é a melhor opção para uma aplicação M2M. Com o intuito de posicionar o leitor no âmbito mais geral do tema abordado neste trabalho, pretende-se: definir os conceitos de Internet das Coisas e M2M, além de descrever uma abordagem da arquitetura de cada um deles, bem como suas camadas e os componentes que as integram; descrever a comunicação dentro do contexto da tecnologia M2M; apresentar características do CoAP e do MQTT. Para a análise do funcionamento dos protocolos, serão desenvolvidas duas aplicações

M2M (cada uma utilizando um dos protocolos estudados), executando sob o sistema operacional Linux.

A metodologia utilizada neste trabalho constitui-se de pesquisa bibliográfica, composta tanto de artigos científicos publicados em jornais de tecnologia disponíveis na Internet que remetem ao tema Internet das Coisas, nos quais são descritos alguns conceitos, análise do cenário atual e possível futuro e também de livro publicado sobre as comunicações nas tecnologias M2M voltado a uma abordagem para sistemas.

2. INTERNET DAS COISAS

O ser humano é um indivíduo de percepção limitada, afinal de contas, tem apenas cinco sentidos. A todo o momento ocorrem diversos eventos capazes de afetar diretamente as pessoas em um determinado ambiente. Por causa desta limitação, sozinhos os humanos não são capazes de prever ou compreender as circunstâncias em que alguns destes eventos ocorreram. Por isso o ser humano inventou instrumentos que ampliam seu campo de visão e o ajudam a entender o ambiente a sua volta como termômetros, odômetros, hidrômetros, sismógrafos...

Além de expandir a percepção, o homem conseguiu, com suas invenções, grandes avanços no campo das comunicações e assim diminuir significavelmente a distância entre as pessoas. A Internet é, sem dúvida, a tecnologia que mais evoluiu e está evoluindo, no que se refere à comunicação das pessoas não importando quantos quilômetros separem umas das outras. Criada em meados da década de 1950, a Internet deixou de ser uma rede estritamente militar para ganhar as universidades e posteriormente, interligar computadores pessoais (PC) no mundo todo.

O advento da tecnologia sem fio trouxe mobilidade à rede de maneira que abriu precedentes para explorar a possibilidade de adicionar outros dispositivos à rede, que não fossem exclusivamente PCs. Assim, nestes últimos anos a Internet passou a conectar outros dispositivos (como *smartphones* e *tablets*), aumentando consideravelmente a população de nós na internet. E é nesse cenário que surge o paradigma de *Internet of Things* (IoT).

Segundo BANDYOPADHYAY et al. (2011, p. 94), “coisas” são dispositivos (físicos ou virtuais) que possuem identidades, atributos e personalidades virtuais e são capazes de utilizar interfaces inteligentes.

Conforme citado anteriormente, existem diversos objetos criados pelo homem com intuito de fornecer informações que não são possíveis de perceber diretamente. Contudo, estas informações são colhidas de forma mecânica e cabe ao indivíduo que coletou tratá-las, armazená-las e/ou retransmiti-las a algum destino.

Tendo isso em mente, é possível verificar o que escrevem alguns autores, sobre IoT:

> Internet das Coisas integra um grande número de tecnologias e visiona uma variedade de coisas ou objetos ao nosso redor que, através de esquemas únicos de endereçamento e protocolos de comunicação padrão, são capazes

> de interagir uns com os outros e cooperar com seus vizinhos para atingirem objetivos comuns (ATZORI et al., 2012, p. 3594).
>
> Internet das Coisas refere-se a uma maneira singular de endereçar objetos e suas representações virtuais em uma estrutura similar à Internet. Tais objetos podem encadear informações sobre si mesmos ou podem transmitir dados em tempo real de sensores sobre seu estado ou outras propriedades úteis associadas a esse objeto (AGGARWAL, 2013, p.383).
>
> Internet das coisas é uma combinação de Internet do futuro e computação ubíqua. Ela demanda interações com sensores heterogêneos, agregadores, atuadores e um diversificado domínio no contexto de aplicações conscientes, preservando segurança. (BANDYOPADHYAY et al., 2011, p. 94).
>
> Interconexão de dispositivos sensores e atuadores provendo a habilidade de compartilhar informações através de plataformas por meio de um framework unificado, desenvolvendo um cenário operacional comum possibilitando aplicações inovadoras. (GUBBI et al., 2013, p. 1645).

A Internet das Coisas surge então como uma rede de nós composta na sua maioria por objetos comuns (isto é, sem recursos computacionais), mas que possui recursos sensoriais ou de medição, por exemplo, que podem ser disponibilizados/compartilhados com seus vizinhos. Uma vez que estes recursos são capazes de circular na rede, eles podem ser utilizados por uma aplicação para armazenamento em algum local, exibição em uma tela, ou retransmissão para um computador a quilômetros de distância. Tudo isso sem que o usuário final interfira no sensor/medidor, na aplicação ou na transmissão dos dados, pois tudo que irá interessar a ele no final das contas é o dado obtido. Isso remete ao conceito de Computação Ubíqua.

2.1. Computação Ubíqua

O termo computação ubíqua foi criado em 1988 por Mark Weiser e abordado por ele em publicações nos anos que seguiram. Para Weiser (1991), computadores fariam parte da vida das pessoas ajudando elas nas tarefas mais simples do dia a dia, em qualquer parte; ou seja, integrar o computador de forma que sua presença para as pessoas fosse mais trivial do que seu objetivo. Weiser ainda afirma que as tecnologias mais profundas são aquelas que desaparecem, isto é, que se envolvem na vida das pessoas de tal forma que se torna imperceptível. Neste mesmo artigo inclusive, ele descreve um cenário onde uma mulher

usufrui de diversos recursos que automatizam e simplificam suas tarefas, como por exemplo, um espelho retrovisor que ajuda a encontrar vagas para estacionar.

Atualmente, a computação ubíqua é uma realidade. Nem todo computador é um PC: temos diversos dispositivos como celulares, *tablets* e *smartphones*, os quais tem capacidade de processamento muito maior que computadores pessoais tinham no inicio do século XXI, e que graças às tecnologias sem fio podem estar conectados à Internet ou entre si a todo o momento, e muitas pessoas usam eles sem a noção de que na verdade, são computadores. A Internet das Coisas aumenta ainda mais a gama de dispositivos que podem se interconectar, englobando objetos do dia a dia para que troquem informações de forma que ajude as pessoas nas tarefas mais corriqueiras, ao mesmo tempo em que transforma o computador em algo "invisível".

2.2. Aplicações IoT para o Mercado

Há vários setores na sociedade que se beneficiariam com a comunicação entre dispositivos. Uma vez que por mais simples que sejam estes dispositivos eles são capazes de comunicar-se com outros ou até com servidores, pode-se pensar em várias aplicações possíveis que poderiam ajudar a melhorar o ambiente em que as pessoas vivem. Fazendo uma analogia, é como se de repente, as coisas ao redor ganhassem bocas e ouvidos tendo a capacidade de passar informações que antes não podiam ser coletadas por seres humanos ou dava muito trabalho fazê-lo. Alguns destes setores são apresentados por BORSWARTHICK, et. al. (2012), tais como: transportes, assistência médica e energia.

Dentro do escopo de aplicações para transporte pode-se citar o rastreamento de ativos, que tem como função rastrear bens em veículos, caminhões ou outras formas de transporte. No tocante a pedágios, é possível realizar a cobrança automática em praças no momento em que o veículo passa. Acerca da navegação, é possível prover informações sobre pontos específicos nos arredores com base em critérios de busca, por exemplo, ao passar por uma cidade desconhecida pode-se encontrar uma pousada mesmo que não exista anúncio visível, pois se um estabelecimento pode 'falar' na rede, uma aplicação pode encontrá-lo e marcar sua localização para o usuário.

No segmento de assistência médica destacam-se as aplicações de monitoramento remoto do paciente, que é responsável por colher informações como pressão arterial,

frequência cardíaca, níveis de glicose entre outras. Destaca-se também o rastreamento de ativos, que abrange uma gama de objetos indispensáveis para um hospital como macas, cadeiras de rodas ou bombas de infusão, uma vez que saber a localização destes objetos em um hospital pode fazer uma grande diferença para salvar uma vida.

No tocante à energia é possível pensar em aplicações no âmbito da medição inteligente como a coleta automática dos níveis de consumo, status e diagnóstico de dispositivos de medição de energia.

Na verdade, quando se fala em controle de consumo, pode-se pensar também em recursos naturais onde se poderia aplicar IoT além de energia elétrica. Como água, por exemplo: uma aplicação capaz de medir os níveis de consumo de água em uma residência que notifica quando estes níveis estão muito altos, por exemplo, ajudaria a combater o desperdício ao mesmo tempo em que pode prevenir o prejuízo financeiro para o responsável além de contribuir na preservação dos recursos hídricos.

Enfim, ao pensar em aplicações para IoT um imenso horizonte se abre diante dos olhos, porque uma vez que ela é composta de objetos do dia a dia, é possível pensar em inúmeras atividades do dia a dia que uma aplicação ajudaria a desempenhar.

2.3. Arquitetura IoT

Como Internet das Coisas encontra-se atualmente em um plano mais conceitual, não existe uma arquitetura totalmente definida. O que se tem é um consenso de alguns profissionais das áreas de internet e comunicação que observaram IoT de um ponto de vista estrutural, separando o conceito em uma arquitetura de camadas. Neste capítulo serão apresentadas duas arquiteturas abordadas para Internet das Coisas.

2.3.1. Arquitetura de três camadas

Alguns autores como Xiadong e Jidong (2010) e Miao et al. (2010) defendem que Internet das Coisas possui uma arquitetura de três camadas. De acordo com Miao et al. (2010, p. 484), a arquitetura de três camadas da Internet das Coisas é amplamente conhecida. Ela é composta por: camada de Percepção (*Perception Layer*), camada de Rede (*Network Layer*), e camada de Aplicação (*Application Layer*), conforme mostra a Figura 1.

Figura 1: Arquitetura de três camadas para IoT

Fonte: Miao et al. (2010)

- **Camada de Percepção:** Como o nome sugere, esta camada é responsável por toda a parte 'sensorial' na Internet das Coisas; responsável por identificar objetos e coletar informações. Ela abrange as "coisas" em IoT, como *tags* RFID, sensores, atuadores, câmeras, GPS.
- **Camada de Rede:** Transmite e processa toda a informação recebida dos objetos;
- **Camada de Aplicação:** Aqui é onde as informações coletadas pela camada de percepção e transmitidas pela camada de rede serão, de fato, utilizadas;

2.3.2. Arquitetura de cinco camadas

Enquanto a arquitetura anterior abortava apenas um nível mais técnico da IoT, a arquitetura de cinco camadas busca analisar também a rede de comunicações envolvendo Internet das Coisas. Segundo MIAO et al. (2010, p. 485), "[...] diferente da Internet, que não pode ser gerenciada ou controlada, a Internet das Coisas precisa ser gerenciada e operada". Então, ao pensar na estrutura de comunicação da IoT, foi elaborada uma arquitetura baseada no modelo TMN[1] (*Telecommunications Management Network*). Esta arquitetura tenta, através da arquitetura da Internet e da estrutura da Rede de Gerenciamento de Telecomunicações (presente no TMN), combinar as duas com características da Internet das Coisas na tentativa de estabelecer uma arquitetura que melhor explique a conotação da IoT. As cinco camadas

[1] Para mais informações sobre TMN acessar http://www.itu.int/en/Pages/default.aspx.

são: Percepção, Transporte, Processamento, Aplicação e Negócio, conforme mostra a Figura 2.

Figura 2: Arquitetura de cinco camadas para IoT

Fonte: Miao et al. (2010)

- **Camada de Percepção:** Assim como no modelo de três camadas, a camada de percepção é encarregada de coletar as informações através de sensores, atuadores, *tags* RFID, etc. Ela também irá transformar estas informações em sinais para transmiti-las na rede.
- **Camada de Transporte:** Como o nome já sugere, a função desta camada é transmitir as informações recebidas da camada anterior através da rede por intermédio dos meios de conexão disponíveis (3G, *Wi-Fi*, *Bluetooth*, infravermelho, etc.). Nesta camada também são encontrados protocolos de endereçamento.
- **Camada de Processamento:** É responsável pelo armazenamento, análise e processamento das informações transmitidas pela camada de transporte.
- **Camada de Aplicação:** Responsável por criar aplicações baseadas nas informações processadas na camada anterior que, ao mesmo tempo, supram as necessidades de negócio para cada segmento no mercado.
- **Camada de Negócio:** Segundo Miao et al. (2010, p. 487), esta camada é como o gerente da Internet das Coisas. Além do lançamento dos aplicativos, esta camada também se preocupa com modelos de negócio e outras questões administrativas como

ROI (*Return of Investment*) e privacidade do usuário.

A arquitetura de cinco camadas expande a visão estrutural da Internet das Coisas ao preocupar-se com a comunicação entre os elementos que a compõem tanto quanto com a conexão entre estes mesmos elementos.

3. *MACHINE-TO-MACHINE* (M2M)

M2M é um acrônimo para *Machine-to-Machine*. Segundo BORSWARTHICK et al. (2012, p. 2), M2M possui um escopo elástico e com limites pouco definidos. Ele define *Machine-to-Machine* da seguinte forma:

O papel de M2M é estabelecer as condições que permitam um dispositivo de trocar informações (bidirecionalmente) com uma aplicação através de uma rede de comunicação, de forma que o dispositivo e/ou aplicação possam agir como a base para esta troca de informações.

M2M engloba dispositivos específicos com a capacidade de (pelo menos) coletar dados em um determinado ambiente. Estes dispositivos transmitem os dados coletados para uma aplicação por meio de uma conexão (cabeada ou não cabeada). Ocasionalmente, estes dispositivos 'coletores' (geralmente sensores) podem ser incapazes de comunicar-se diretamente com aplicação, fazendo-se necessária a utilização de outro dispositivo que irá agir como um intermediário na comunicação transmitindo os dados recebidos até o outro lado (um gateway). É importante ressaltar que durante este processo não há intervenção de um ser humano, isto é, não há uma requisição prévia feita por um usuário, que será recebida pelo dispositivo sensor para ser processada em seguida.

Sendo assim, pode-se definir M2M como um conjunto de tecnologias que objetivam estabelecer uma comunicação (cabeada ou não-cabeada) entre dispositivos sensores com capacidade (limitada ou nula) de transmitir informações coletadas para uma (ou várias) aplicação sem que um ser humano participe da comunicação.

3.1. Comparação entre M2M e IoT

Os termos M2M e Internet das Coisas se confundem entre si, uma vez que as relações entre eles não estão devidamente definidas. No que se refere a estas relações, há varias opiniões diferentes expressas por alguns indivíduos do meio da Tecnologia da Informação. Nesta seção são apresentados alguns destes pontos de vista.

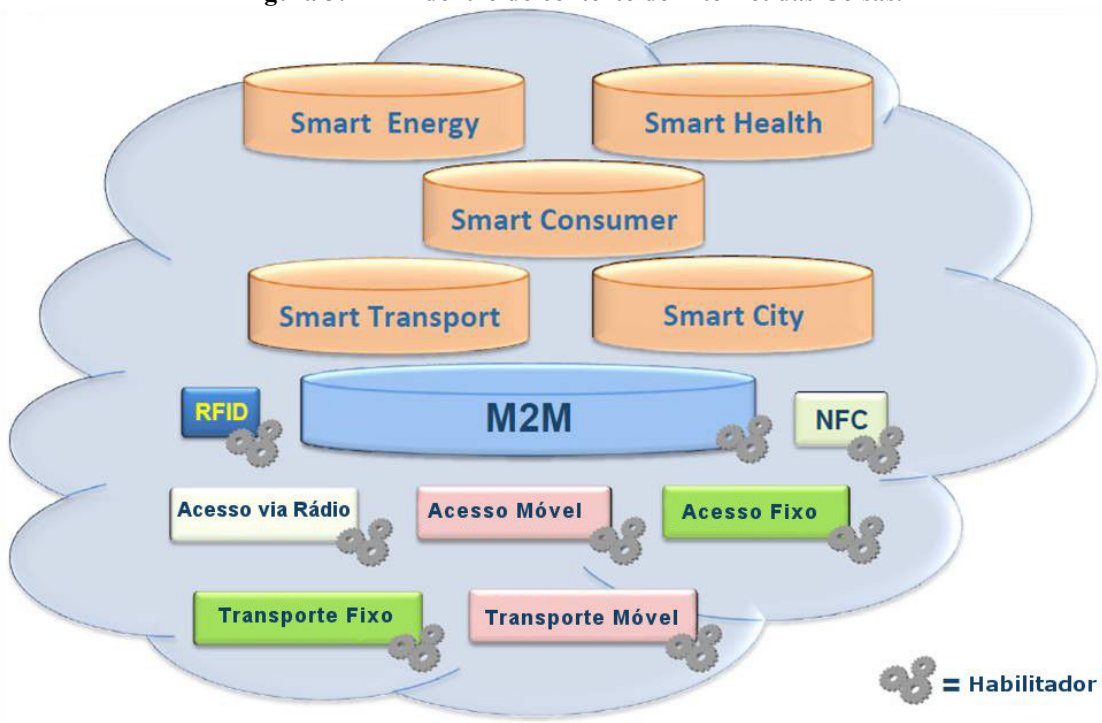
Para BORSWARTHICK et al. (2012, p. 5) apesar de haver sobreposições entre os termos, um não pode ser considerado um subconjunto do outro, pois “existem áreas que são particularmente especificas para cada um deles”, como por exemplo, o fato algumas ‘coisas’

em IoT não se tratarem necessariamente de objetos que estabelecem um ‘relacionamento M2M’ com a aplicação, uma vez que são passivos e não transmitem as informações; elas apenas são lidas por um outro dispositivo (*tags* RFID).

Segundo POLSONETTI (2014), M2M é uma tecnologia precursora à Internet das Coisas, pois enquanto M2M possui soluções baseadas em comunicações ponto-a-ponto usando dispositivos embarcados com o objetivo de reduzir custos de gerenciamento de recursos através de diagnósticos remotos, atualizações ou suporte técnico, as soluções de IoT envolvem um acesso muito mais amplo, acomodando sensores passivos, entre outros objetos, visando melhorias não só na produção e prestação do serviço das empresas, mas também no modelo de negócio.

Comunicações *Machine-to-Machine* podem ser vistas como uma precursora de Internet das Coisas, pois ela consegue colocar em prática alguns cenários parecidos com aqueles que se imaginam para IoT futuramente, da mesma forma que IoT busca englobar e estender o elementos presentes nas comunicações M2M aplicando, além da troca de informação, conceitos como *Big Data* e *Smart Cities*. A Figura ilustra a tecnologia M2M no contexto de Internet das Coisas.

Figura 3: M2M dentro do contexto de Internet das Coisas.



Fonte: ETSI (2012)

3.2. Arquitetura M2M

Em Janeiro de 2009, o *European Telecommunications Standards Institute* (ETSI) criou um comitê técnico (ETSI TC M2M) com o objetivo de prover uma visão fim-a-fim da arquitetura da comunicação M2M através de padrões, os quais não são totalmente desenvolvidos pelo Comitê, mas também por outras organizações (BOSWARTHICK et al., 2012, p. 58).

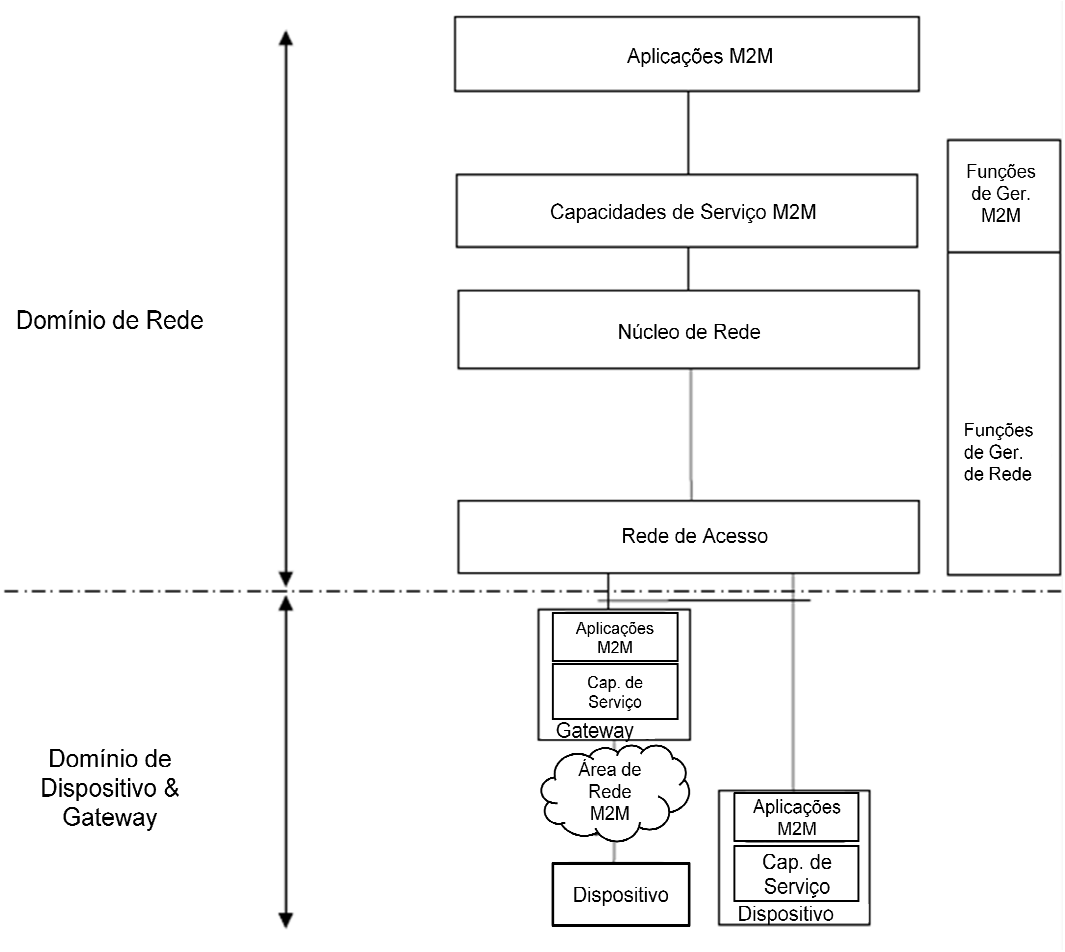
Em uma de suas *Technical Specifications* (TS 102690), o comitê (agora nomeado para *SmartM2M*[2]) define uma arquitetura de alto nível para a comunicação M2M, que é apresentada na Figura 4 (ETSI, 2013 a).

Como podemos observar, a arquitetura divide-se em dois domínios: Dispositivo & Gateway e Rede. De acordo com o *SmartM2M,* os componentes destes domínios são descritos da seguinte forma:

- Domínio de dispositivo & gateway
    - Dispositivo M2M: um dispositivo que roda uma ou mais aplicações M2M usando as capacidades de serviço M2M;
    - *Gateway*: Objeto que roda aplicações M2M usando as capacidades de serviço M2M e que atua como um *proxy* entre dispositivos M2M e o domínio de rede.
    - Área de Rede M2M: provê a conexão entre os dispositivos e os *gateways*;
- Domínio de Rede
    - Rede de Acesso: permite que os dispositivos e os *gateways* M2M se comuniquem com o núcleo de rede;
    - Núcleo de Rede: responsável por prover funções que garantem o funcionamento da rede como conectividade via IP e potencialmente outros meios de conectividade, bem como funções de controle de serviço e rede, interconexão (com outras redes) e *roaming*;

[2] Informações sobre a mudança de nome do comitê técnico podem ser encontradas em http://portal.etsi.org/TBSiteMap/SmartM2M/ActivityReport.aspx

Figura 4: Arquitetura M2M

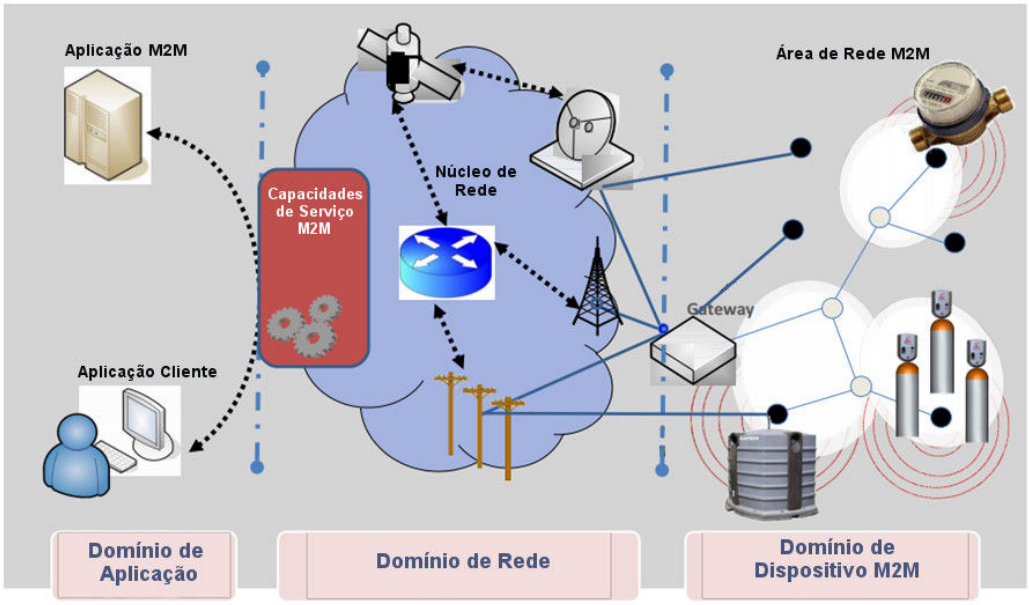


Fonte: ETSI TC M2M (2013)

- Capacidades de Serviço M2M**:** provêm funções M2M que são compartilhadas por diferentes aplicações. Além de exporem funções através de um conjunto de interfaces abertas. Também utilizam as funcionalidades do núcleo de rede, e simplificam e aperfeiçoam o desenvolvimento e lançamento da aplicação escondendo as especificações da rede;
- Aplicações M2M: rodam a lógica de serviço e usam as capacidades de serviço que são acessíveis por meio de uma interface aberta;
- Funções de Gerenciamento de Rede: agrupam todas as funções necessárias para gerenciar núcleo de rede e a rede de acesso;
- Funções de Gerenciamento M2M: consiste nas funções necessárias para gerenciar as capacidades de serviço dentro do domínio de rede;

O ETSI também possui uma versão mais simplificada desta arquitetura, que dá conta de alguns elementos mais importantes da arquitetura de alto nível, alem de definir um novo domínio: o Domínio de Aplicação. Esta versão está representada na Figura 5.

Figura 5: Versão simplificada da arquitetura de Alto Nível



Fonte: ETSI TC M2M (2012)

4. PROTOCOLOS DE COMUNICACÃO

Na Internet 'tradicional' existem diversos protocolos de comunicação responsáveis por gerenciar a transferência de dados em uma rede que conecta dois ou mais computadores. Para citar alguns exemplos: HTTP, FTP e SFTP. Quando se fala em comunicação entre dois ou mais dispositivos (ou um conjunto de aplicações) conectados em uma rede também surge a necessidade de pensar em um protocolo que gerencie esta comunicação, isto é, a troca de mensagens e/ou dados entre os elementos que compõe esta rede de objetos de forma eficiente considerando as características e limitações impostas pelo ambiente. Neste cenário, surgem dois protocolos de transferência que podem ser utilizados em ambientes restritos (capacidade computacional limitada, rede com perdas...): o *Messaging Queue Telemetry Transport* (MQTT) e o *Constrained Application Protocol* (CoAP).

4.1. MQTT

O MQTT foi criado em meados de 1999 por Andy Stanford-Clark (IBM) e Arlen Nipper (*Eurotech*). Trata-se de um protocolo de mensagens baseado na arquitetura *publish*/*subscribe*, voltado para dispositivos restritos e redes inseguras, com baixa largura de banda e alta latência. Segundo a página *web* do projeto[3], os princípios do design são minimizar os requerimentos de recursos de dispositivo e de largura de banda tentando garantir confiabilidade e garantia de entrega.

Atualmente na versão 3.1, a especificação do MQTT (IBM; Eurotech, 2010) apresenta uma série de características do protocolo, algumas delas listadas abaixo:

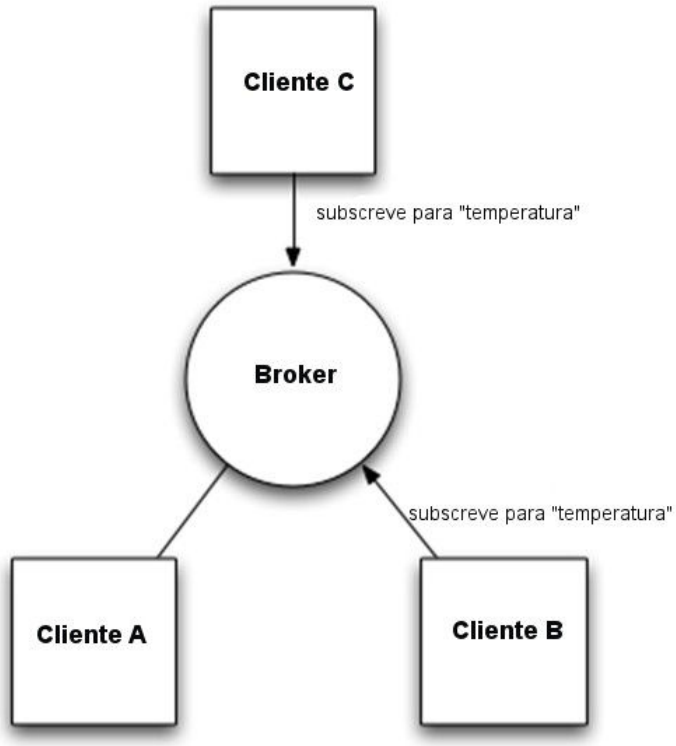
- Uso de TCP/IP para fornecer conectividade;
- Pequena sobrecarga de transporte e trocas minimizadas de protocolos para reduzir tráfego na rede;
- Mecanismo que notifica partes interessadas quando um cliente se desconecta da rede anormalmente;

Segundo Jaffey (2014), o protocolo segue o modelo cliente/servidor. Os dispositivos sensores são clientes que se conectam a um servidor (chamado de *broker*) usando TCP. As

[3] Disponível em: http://mqtt.org/

mensagens a serem transmitidas são publicadas para um endereço (chamado de tópico), que inclusive, assemelha-se a uma estrutura de diretórios em um sistema de arquivos, por exemplo, casa/quarto2/temperatura. Clientes por sua vez podem se subscrever para vários tópicos, tornando-se assim capazes de receber as mensagens que outros clientes publicam neste tópico. A Figura 6 mostra uma rede conectando três clientes com um *broker* central.

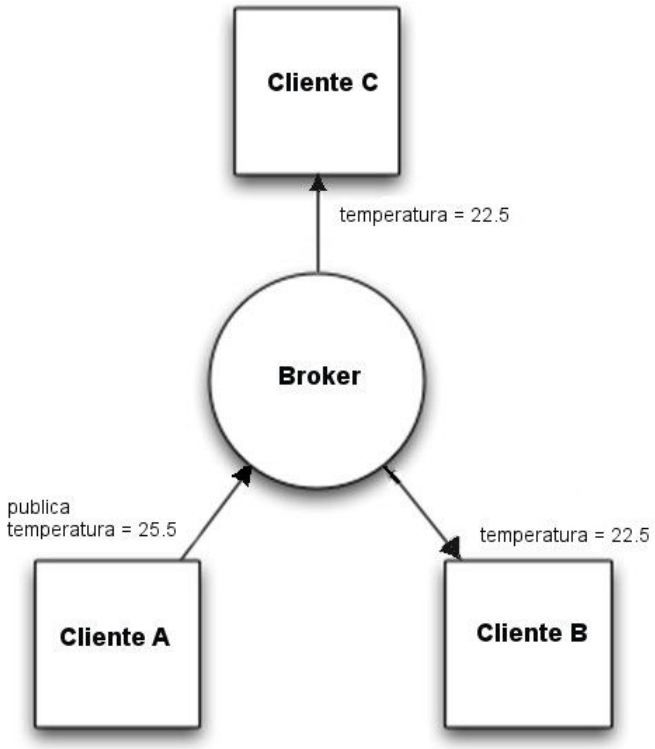
Figura 6: Subscrição de tópico ao *broker*



Fonte: Jaffey (2014)

Conforme a Figura 6, todos os clientes conectam-se ao *broker*, e os clientes B e C subscrevem para o tópico temperatura. Posteriormente, o cliente A publica o valor '22.5' para o tópico temperatura. O *broker* então encaminha a mensagem para os clientes subscritos para aquele tópico, possibilitando assim, que os clientes comuniquem-se um-a-um, um-para-muitos ou muitos-para-muitos. O processo de publicação da mensagem pode ser visto na Figura 7.

Figura 7: Publicação de mensagem no *broker*



Fonte: Jaffey (2014)

4.1.1. Formato da Mensagem

Cada uma das chamadas mensagens de comando MQTT possui um cabeçalho fixo composto de dois bytes, onde o primeiro byte contém o campo que identifica o tipo da mensagem e também campos marcadores (DUP, nível QoS e RETAIN). A Figura 8 mostra o formato do cabeçalho fixo das mensagens.

Figura 8: Cabeçalho fixo de uma mensagem MQTT

bit	7	6	5	4	3	2	1	0
Byte 1	Tipo da Mensagem				Flag DUP	Nível QoS		RETAIN
Byte 2	Largura restante							

Fonte: IBM & Eurotech (2010)

O tipo da mensagem ocupa do bit 7 ao 4 do primeiro byte do cabeçalho fixo e é representado por um valor não assinado. A lista de tipos definida na versão 3.1 do MQTT é descrita na Tabela 1.

Os quatro bits restantes do primeiro byte dividem-se em quatro campos que servem de marcadores para indicar preferências definidas antes do envio da mensagem. São eles:

- DUP: acrônimo relativo à *Duplicate delivery* (entrega duplicada), este marcador ocupa o bit 4 e é ativado quando o cliente ou o servidor tentam reenviar mensagens do tipo *PUBLISH*, *PUBREL*, *SUBSCRIBE* ou *UNSUBSCRIBE* (ver Tabela 1) que tenham QoS (*Quality of Service*) maior que 0 e requeiram *acknowlegment* (*ACK*).
- QoS: Este marcador ocupa os dois penúltimos bits e indica o nível de garantia da entrega de uma mensagem *PUBLISH*. Os níveis de QoS são mostrados na Tabela 2.
- *RETAIN*: Quando um cliente envia uma mensagem *PUBLISH* ao servidor com este marcador ativado, ela deve ser retida no servidor mesmo depois de ser entregue aos assinantes. No evento de uma nova subscrição a um tópico, a última mensagem retida para este tópico deve ser enviada para o novo assinante caso este marcador esteja ativado.

Tabela 1: Tipos de mensagem.

Mnemônico	Enumeração	Descrição
Reservado	0	Reservado
CONNECT	1	Requisição de conexão do cliente ao servidor
CONNACK	2	*ACK* de conexão
PUBLISH	3	Publicação de mensagem
PUBACK	4	*ACK* de publicação
PUBREC	5	Publicação recebida (garantia de entrega parte I)
PUBREL	6	Publicação liberada (garantia de entrega parte II)
PUBCOMP	7	Publicação completa (garantia de entrega parte III)
SUBSCRIBE	8	Requisição de subscrição do cliente
SUBACK	9	*ACK* de subscrição
UNSUBSCRIBE	10	Requisição de cancelamento de subscrição do cliente
UNSUBACK	11	*ACK* de cancelamento de subscrição
PINGREQ	12	Requisição *PING*
PINGRESP	13	Resposta *PING*
DISCONNECT	14	Cliente desconectando
Reservado	15	Reservado

Fonte: IBM & Eurotech (2010)

A largura restante do cabeçalho fixo (byte 2) é usada para representar nada mais do que a quantidade de bytes remanescentes na mensagem. Incluindo dados do cabeçalho variável e do *payload.*

Cabeçalho variável é um componente presente em alguns tipos de mensagem MQTT e está localizado entre o cabeçalho fixo e o *payload.* Usado principalmente nas mensagens *CONNECT*, este cabeçalho possui dois campos para nome e versão do protocolo, respectivamente e mais uma série de marcadores que definirão algumas diretivas para a conexão entre cliente e servidor.

Tabela 2: Níveis de QoS

Valor QoS	Bit 2	Bit 1	Descrição		
0	0	0	Até uma vez	Disparar e esquecer	≤ 1
1	0	1	Ao menos uma vez	Entrega com *ACK*	≥ 1
2	1	0	Uma vez	Entrega garantida	1
3	1	1	Reservado		

Fonte: IBM & Eurotech (2010)

O *payload* pode armazenar diferentes tipos de informação, tudo vai depender do tipo da mensagem transmitida: *CONNECT* (irá conter ID do cliente), *SUBSCRIBE* (contém tópicos que o cliente pode subscrever e/ou nível QoS) ou *SUBACK* (lista de níveis de QoS garantidos pelo servidor).

Desde março de 2013, o MQTT está em processo de padronização na OASIS (um consórcio sem fins lucrativos que conduz o desenvolvimento, convergência e adoção de padrões abertos para a sociedade da informação). A última noticia do comitê técnico da OASIS[4] para o MQTT dá conta de que o protocolo encontra-se em período de avaliação como um *Candidate OASIS Standart* (COS), porém sem atualizações mais recentes.

4.2. CoAP

O CoAP foi criado por um grupo de trabalho do *Internet Engineering Task Force* (IETF) chamado *Constrained RESTful Environments* (CoRE), composto atualmente por Carstem Bormann, Andrew McGregor e Barry Leiba. O grupo iniciou suas atividades em março de 2010 com objetivo de prover um *framework* para aplicações que manipulam recursos simples localizados em dispositivos interligados em redes limitadas, incluindo desde

[4] Mais informações em: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt

aplicações que monitoram sensores de temperatura e medidores de energia, até controle de atuadores como interruptores ou trancas eletrônicas, e também aplicações que gerenciam os dispositivos que compõem a rede, conforme o IETF (2010). O CoAP trata de uma parte deste *framework*. Sendo assim, a definição do protocolo foi um dos primeiros marcos definidos pelo CoRE. O primeiro *draft* para o CoAP foi publicado no IETF em junho de 2010. Nós últimos quatro anos, o documento teve ao todo dezoito versões até que recentemente (junho de 2014), tornou-se um RFC.

De acordo com o RFC 7252 (IETF, 2014, p. 1), o CoAP é um protocolo de transferência voltado para nós e para redes restritas (considerando nós com pequena quantidade de memória RAM e redes em que a taxa de perda de pacotes é alta). O protocolo foi projetado para aplicações M2M como, por exemplo, *smart energy* e automação residencial. Ele define quatro tipos de mensagens: *Confirmable*, *Non-confirmable*, *Acknowledgment* e *Reset* (RFC 7252, 2014, p. 8).

Confirmable (CON), como o nome sugere, são mensagens que precisam ser confirmadas no destino. Assim, quando não há perda de pacotes, cada mensagem deste tipo resulta em uma mensagem do tipo *Acknowledgment* ou *Reset.*

Mensagens *Non-confirmable* (NON), não necessitam de confirmação de recebimento. Esta característica é útil no caso de uma aplicação que recebe leituras constantes de um sensor de temperatura em um espaço muito curto de tempo, onde a perda de uma ou outra mensagem não é motivo para preocupações.

Acknowledgment são mensagens que confirmam o recebimento de uma mensagem *Confirmable.* É importante ressaltar que por si só, uma mensagem *ACK* não indica sucesso ou falha de nenhuma requisição encapsulada na mensagem *Confirmable*.

Mensagens do tipo *Reset* indicam que outra mensagem (CON ou NON) foi recebida, mas por falta de algum contexto ela não pôde ser devidamente processada. Ela pode ocorrer no caso de algum dispositivo ter reiniciado e a mensagem enviada não foi devidamente interrompida.

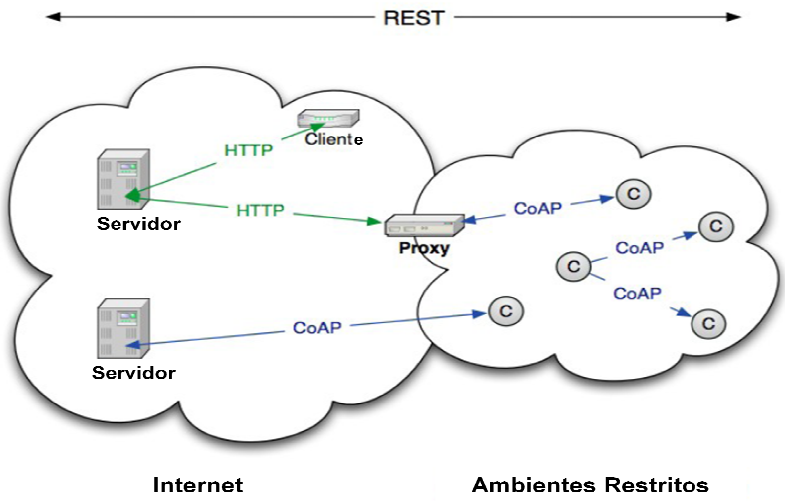
O CoAP provê um modelo de interação requisição/resposta entre os *endpoints* das aplicações, suporta descoberta embutida de serviços e recursos e inclui conceitos chave da *Web* como por exemplo *Uniform Resources Identifiers* (URI's) e tipos de mídia comuns na

Internet. Além disso, o protocolo foi projetado para interagir com o HTTP para integração com a *Web*.

Abaixo estão algumas características descritas no RFC 7252:

- Protocolo *Web* que cumpre com os requerimentos M2M em ambientes restritos;
- Troca de mensagens assíncrona;
- Suporte à URI e *Content-Type*;
- Capacidades simples de *proxy* e *caching*;
- Mapeamento HTTP que permite que *proxies* possam prover acesso aos recursos do CoAP via HTTP de maneira uniforme.
- Interligação segura para *Datagram Transport Layer Security* (DTLS);
- *Binding* em UDP com confiabilidade opcional suportando requisições tanto *unicast* quanto *multicast*;
- Suporte aos métodos *GET*, *POST*, *PUT*, *DELETE*;

Figura 9: Arquitetura CoAP



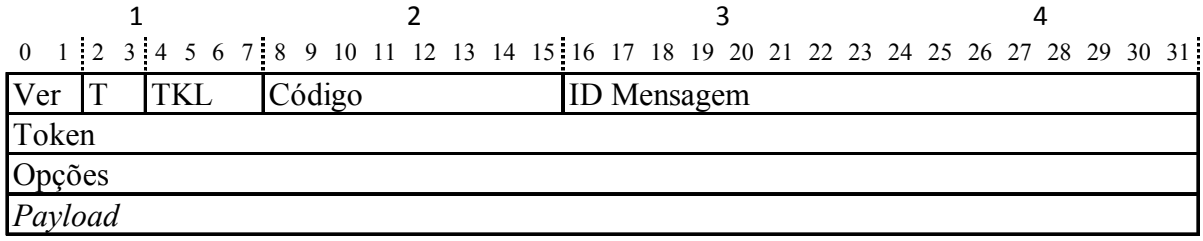
Fonte: Shelby (2013)

A Figura 9 ilustra a arquitetura CoAP em uma perspectiva de alto nível. Um dos objetivos do CoRE também consiste em adequar a arquitetura REST (*Representational State Transfer,* proposta por Fielding (2000)) para ambientes restritos (nós e rede). O CoAP foi desenvolvido de acordo com esta arquitetura, logo pode ser considerado como um protocolo *RESTful,* de acordo com SHELBY (2013).

4.2.1. Formato da Mensagem

CoAP é baseado na troca de mensagens compactas que são transportadas sobre UDP (*User Datagram Protocol*). Suas mensagens estão codificadas em um formato binário com um cabeçalho de 4 bytes, seguido por um Token de largura variável (de 0 a 8 bytes) . Após o token pode não haver nenhuma ou uma série de opções do CoAP no formato TLV (*Type-Length-Value*), que são opcionalmente seguidas por um *payload*, ocupando o resto do datagrama. A Figura 10 mostra o formato da mensagem CoAP.

Figura 10: Formato da Mensagem CoAP



Fonte: IETF (2014)

Os campos que formam o cabeçalho do datagrama são definidos pelo RFC da seguinte maneira:

- **Versão (Ver)**: inteiro não assinado de dois bits que indica o número correspondente à versão do CoAP;
- **Tipo (T)**: inteiro não assinado de dois bits que indica se o tipo da mensagem, que pode variar entre *Confirmable* (0), *Non-confirmable* (1), *Acknowlegment* (2) ou *Reset* (3);
- **Largura do Token (TKL)**: inteiro não assinado de quatro bits usado para indicar a largura variável do Token (até 8 bytes).
- **Código**: inteiro não assinado de oito bits que caracteriza o tipo da mensagem, podendo indicar um método de requisição (Tabela 3) ou um código de resposta (Tabela 4);
- **ID da mensagem**: inteiro não assinado de 16 bits usado para detectar duplicação de mensagens e também para comparação de mensagens do tipo *ACK*, por exemplo;

O *Token* é utilizado para relacionar requisições e respostas. Ele é gerado no cliente e transportado junto com a requisição. O servidor deve então ecoar este token na resposta correspondente.

Conforme citado anteriormente, seguindo o cabeçalho, o Token e as opções (uso opcional), vem o *payload*, que é um campo opcional. Quando a mensagem possui *payload* ele deve ser indicado por um marcador de um byte (*Payload Marker*) que indica o fim das opções e o início da carga. A ausência do marcador significa um *payload* de largura 0 bytes, enquanto a presença de um marcador seguido por um *payload* de 0 bytes deve ser processado como um erro no formato da mensagem.

Tabela 3: Códigos dos métodos CoAP

Código	Nome
0.01	*GET*
0.02	*POST*
0.03	*PUT*
0.04	*DELETE*

Fonte: IETF (2014)

Tabela 4: Códigos de resposta CoAP

Código	Nome
2.01	Criado
2.02	Deletado
2.03	Válido
2.04	Alterado
2.05	Conteúdo
4.00	*Bad Request*
4.01	Não Autorizado
4.02	*Bad Option*
4.03	Proibido
4.04	Não Encontrado
4.05	Método Não Permitido
4.06	Inaceitável
4.12	Precondição Falhou
4.13	Entidade de Requisição Muito Grande
4.15	*Content-Format* não Suportado
5.00	Erro Interno no Servidor
5.01	Não Implementado
5.02	*Bad Gateway*
5.03	Serviço Indisponível
5.04	*Gateway Timeout*
5.05	*Proxying* Não Suportado

Fonte: IETF (2014)

5. DESENVOLVIMENTO

Uma vez que o objetivo geral deste trabalho foi o de realizar um comparativo entre o MQTT e o CoAP, foram desenvolvidas duas aplicações que usam estes protocolos de forma que seja gerado tráfego de dados na rede para que o mesmo possa ser capturado e analisado por um software de captura de pacotes afim de, à partir da análise dos resultados, apontar qual a melhor opção entre os dois para aplicações M2M. Nas seções seguintes são explicados alguns pontos envolvendo tanto as aplicações e o ambiente em que elas funcionam.

5.1. Ambiente

Apesar de este trabalho citar a capacidade dos protocolos trabalharem bem em ambientes reais com conectividade e processamento restritos, não foi possível reproduzir estes mesmos ambientes no desenvolvimento. Por isso, foi criado um ambiente controlado para que as aplicações pudessem produzir dados para análise.

Assim, ambas as aplicações executam em um computador pessoal com sistema operacional Linux (distribuição Fedora versão 20 para arquitetura 64 bits), e todo o tráfego de rede é gerado na interface local da máquina.

Para trabalhar com o MQTT, o ambiente ainda conta com o *mosquitto*, um *broker* criado por Roger Light, que inclusive oferece suporte à subscrição e publicação em tópicos. O CoAP não exigiu nenhuma instalação adicional ao sistema operacional.

5.2. Aplicação

O objetivo das aplicações consiste em simular a transmissão de informações de temperatura de um cliente para um servidor através da rede utilizando os protocolos abordados neste trabalho. Para tanto foram desenvolvidas duas aplicações contemplando este cenário. As funções contempladas pelo par de aplicações serão 1) Simular a informação de temperatura de um cliente (em Celsius, Kelvin ou Fahrenheit) e 2) transmitir essa informação para um servidor (ou outros clientes, no caso do MQTT).

5.3. Programação da aplicação usando MQTT

A aplicação voltada ao MQTT foi codificada em Python, utilizando a biblioteca paho-mqtt-client, mantida pela comunidade Eclipse através do projeto Paho[5]. A Figura 11 mostra o código da aplicação com alguns blocos destacados, os quais são elucidados em seguida.

Figura 11: Aplicação utilizando protocolo MQTT.

publica_temperatura.py

```
1  import paho.mqtt.publish as publish
2  from random import randint
3
4  def celcius2kelvin(val):
5      return val+273
6
7
8  def celcius2fahrenheit(val):
9      return (val*1.8)+32
10
11 while True:
12
13     temperatura = randint(-40,40)
14
15     celsius = str(temperatura)+" oC"
16
17     kelvin = str(celcius2kelvin(temperatura))+" oK"
18
19     fahrenheit= str(celcius2fahrenheit(temperatura))+ " oF"
20
21     dict_celsius = {'topic':'temperatura/celsius','payload':celsius}
22
23     dict_kelvin = {'topic':'temperatura/kelvin','payload':kelvin}
24
25     dict_fahrenheit = {'topic':'temperatura/fahrenheit','payload':fahrenheit}
26
27     list_msg = [dict_celsius,dict_kelvin,dict_fahrenheit]
28
29     publish.multiple(list_msg, hostname="127.0.0.1")
30
31     if(raw_input()):
32         break
```

1 2 3 4

Fonte: Próprio autor.

A aplicação conta com um laço ***while*** que só ira parar quando um caractere for lido do teclado. No bloco nº 1 estão as funções que convertem o valor da temperatura nas escalas Kelvin e Fahrenheit. Uma vez que o ambiente não possui um sensor capaz de medir a

[5] Mais detalhes em http://eclipse.org/paho/

temperatura, a aplicação gera números randômicos inteiros entre -40 e 40 para atribuí-los à variável responsável por armazenar o dado, conforme o bloco nº 2. No bloco nº 3 cada mensagem é transformada em um variável dicionário do Python (***dict*)** composto basicamente da identificação do tópico e do *payload* (devidamente convertidos para as escalas Kelvin e Fahrenheit). Estes dicionários, por sua vez, vão compor uma variável ***list*** que será um dos parâmetros para a função que publica todas as mensagens para o *broker* de uma vez, conforme o bloco nº 4.

Destaca-se que a aplicação faz uso da hierarquia de tópicos fornecida pelo MQTT: Todas as três escalas são subtópicos de temperatura; então subescrevendo ao tópico temperatura utilizando o caractere coringa # (**temperatura/#**) o assinante recebe as mensagens de todos os subtópicos sem distinção.

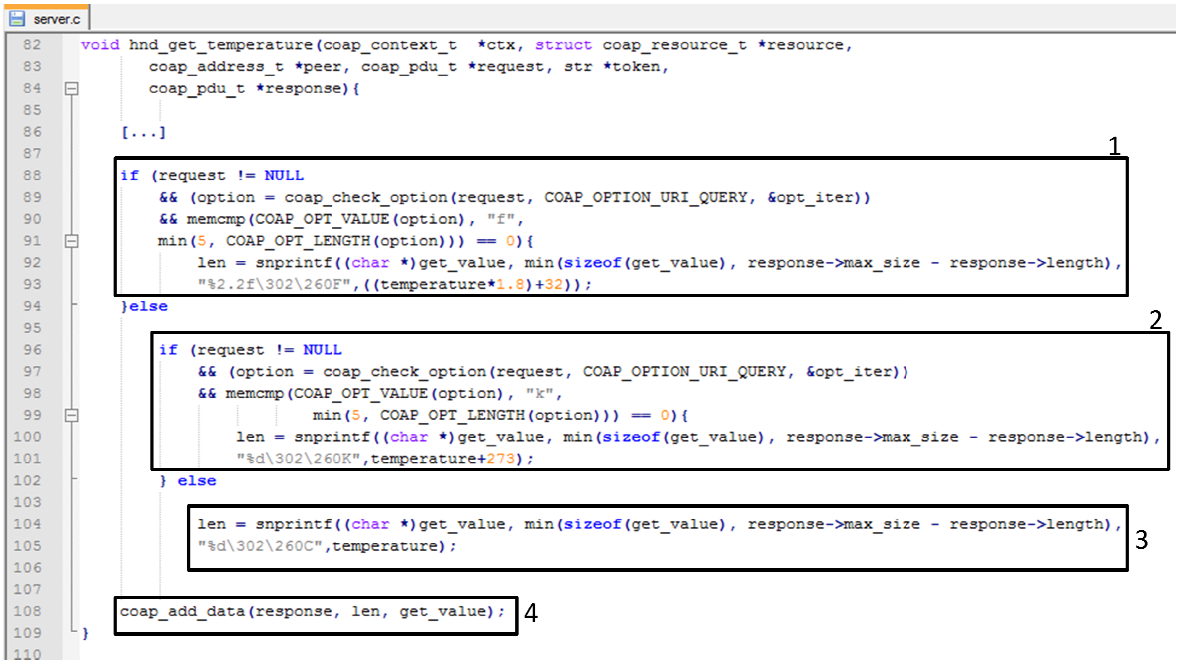
5.4. Programação da aplicação usando CoAP

Para a aplicação voltada ao CoAP foi utilizada a implementação *libcoap*, desenvolvida por BERGMANN (2010) em linguagem C. A implementação disponibilizada por BERGMANN conta com um diretório chamado '*example*' que possui um cliente e um servidor CoAP. Com o intuito de fazer adequações ao cenário proposto, algumas alterações foram feitas no código fonte do servidor, as quais são explicadas a seguir.

O servidor foi codificado para fornecer recursos que podem ser acessados ou alterados pelos métodos *GET*, *PUT*, *POST* ou *DELETE*, conforme descrito no RFC 7252. Assim, tanto o recurso quanto manipulação determinada por um dos métodos deve ser implementada no servidor para que o cliente possa ter acesso a elas.

Como o exemplo disponibilizado no *libcoap* não possuía recurso de temperatura nem manipuladores para os métodos do recurso, conforme proposto no cenário, o código fonte precisou dos seguintes ajustes: criação da função que atua como manipulador do recurso de temperatura para o método *GET* (*hnd_get_temperature*), e o mesmo para o método *PUT* (*hnd_put_temperature*). A Figura 12 traz partes da função para o método *GET*, com alguns blocos de código destacados, os quais são comentados abaixo.

Figura 12: Função que cria o manipulador para o método *GET* da aplicação CoAP.




```
82  void hnd_get_temperature(coap_context_t  *ctx, struct coap_resource_t *resource,
83        coap_address_t *peer, coap_pdu_t *request, str *token,
84        coap_pdu_t *response){
85
86      [...]
87
88      if (request != NULL
89          && (option = coap_check_option(request, COAP_OPTION_URI_QUERY, &opt_iter))
90          && memcmp(COAP_OPT_VALUE(option), "f",
91          min(5, COAP_OPT_LENGTH(option))) == 0){
92              len = snprintf((char *)get_value, min(sizeof(get_value), response->max_size - response->length),
93              "%2.2f\302\260F",((temperature*1.8)+32));
94      }else
95
96          if (request != NULL
97              && (option = coap_check_option(request, COAP_OPTION_URI_QUERY, &opt_iter))
98              && memcmp(COAP_OPT_VALUE(option), "k",
99                        min(5, COAP_OPT_LENGTH(option))) == 0){
100             len = snprintf((char *)get_value, min(sizeof(get_value), response->max_size - response->length),
101             "%d\302\260K",temperature+273);
102         } else
103
104             len = snprintf((char *)get_value, min(sizeof(get_value), response->max_size - response->length),
105             "%d\302\260C",temperature);
106
107
108     coap_add_data(response, len, get_value);
109 }
110
```

Fonte: Próprio autor.

Os blocos de nº 1 a 3 fazem a mesma coisa, que é preparar os componentes do *Protocol Datagram Unity* (PDU), ou seja, o datragama da reposta à requisição propriamente dita. A diferença é que nos blocos nº 1 e 2 é considerada a *query string* da requisição que servirá como opção para recuperar a informação na escala Kelvin ou na Fahrenheit (usando as letras k e f, respectivamente). Já no bloco nº 3 o dado não sofre nenhuma transformação de escala, ou seja, retorna o valor em Celsius.

Independente do tratamento que o dado da temperatura recebe, nos três casos ocorre a composição da largura do dado e do dado propriamente dito, dois componentes imprescindíveis para a resposta da requisição, sendo representados por uma variável do tipo **size_t** chamada *len* e um vetor de **char** chamado *get_value,* respectivamente. A função **sprintf** assim como a **printf** forma uma *string* de acordo com o formato passado por parâmetro, só que ao invés de enviar a *string* para a saída padrão, esta função a armazena em um buffer, também passado por parâmetro.

O valor retornado por esta função é o numero de bytes escritos no *buffer*[6]. Na aplicação, a variável *len* serve para identificar o número de bytes enquanto *get_value* representa o *buffer*, que contém o dado a ser transmitido. Por fim, no bloco nº 4, os dados especificados nos parâmetros são passados à variável *response* na função *coap_add_data*, que como o nome sugere, adiciona as informações ao PDU.

A função *hnd_put_temperature* é mais simples e todo seu conteúdo está ilustrado na Figura 13. O bloco nº 1 destaca a função *coap_get_data* que recupera as informações contidas no PDU da requisição para os ponteiros *size* e *data*. Se o tamanho do dado contido no PDU for zero, o código 400 (*Bad Request*) é adicionado ao cabeçalho da resposta e nada mais acontece (bloco nº2), caso contrário, a variável *temperature* recebe o novo valor (bloco nº 3) concluindo assim a requisição *PUT*.

Figura 13: Função que cria o manipulador para o método *PUT* da aplicação CoAP.

server.c

```
118  void hnd_put_temperature(coap_context_t  *ctx, struct coap_resource_t *resource,
119        coap_address_t *peer, coap_pdu_t *request, str *token,
120        coap_pdu_t *response){
121    size_t size;
122    unsigned char *data;
123    response->hdr->code =
124      temperature ? COAP_RESPONSE_CODE(205) : COAP_RESPONSE_CODE(404);
125
126      coap_get_data(request,&size,&data);                          1
127
128      if(size == 0){
129         response->hdr->code = COAP_RESPONSE_CODE(400);            2
130      }
131      else
132       temperature = atoi((char*)data);                            3
133
134  }
135
```

Fonte: Próprio autor.

Por fim, após a criar os manipuladores foram inseridas as linhas de código que criaram o recurso de temperatura através da função *coap_resource_init*, além de registrar os dois

[6] Conforme descrito em http://www.cplusplus.com/reference/cstdio/sprintf

manipuladores de métodos com a função *coap_register_handler,* conforme ilustra a Figura 14.

Figura 14: Registrando os manipuladores no recurso recém-criado.

server.c

```
354   r = coap_resource_init((unsigned char *)"temperature", 11, 0);
355   coap_register_handler(r, COAP_REQUEST_GET, hnd_get_temperature);
356   coap_register_handler(r, COAP_REQUEST_PUT, hnd_put_temperature);
357
```

Fonte: Próprio autor.

5.5. Testes

Os testes realizados são exclusivamente voltados à transmissão de dados através da rede utilizando as capacidades de cada um dos protocolos estudados, ou seja, eles exploram tanto do envio quanto o recebimento de mensagens dos protocolos. O conteúdo transmitido consiste de uma *string* formada pelo valor que simula a temperatura, juntamente com o símbolo (º) e a letra que identifica a escala (C, K ou F).

Conforme explicado na seção anterior, a aplicação MQTT faz a publicação dos dados continuamente. A outra parte do teste consiste na subscrição aos relativos tópicos através do comando executado em terminal de linha de comando no Linux. Nos testes foram feitas ao todo quatro subscrições, sendo uma para cada escala termométrica e outra para todas as escalas sem separação, as quais devem receber as mensagens simultaneamente.

A forma geral da subscrição é a seguinte: *mosquitto_sub* –t **nome_topico**. Onde mosquitto_sub é o comando de subscrição fornecido pelo *broker mosquitto*, '-t' é a opção que indica que um tópico vai ser subscrito e **nome_topico** é o parâmetro que indica o nome do tópico subscrito.

A aplicação CoAP, por outro lado, não tem o dado que simula a temperatura alterado por um valor atribuído de forma randômica. Ela começa com um valor fixo e só pode ser alterada por uma requisição *PUT*, e deve ser recuperada por meio de uma requisição *GET*. Assim os testes com esta aplicação constituíram uma requisição *PUT* e três requisições *GET*, sendo uma para cada escala termométrica. Com o servidor CoAP rodando na máquina o comando para as requisições *PUT* e *GET* segue a seguinte estrutura:

- *coap_client* –m put –e **dado_enviado** [host][/recurso], onde coap_client é a chamada da aplicação escrita em C, '-m put' indica que o *PUT* será o método da requisição, '-e' indica que um texto será enviado como *payload* através da *string* **dado_enviado**, que contém a informação a ser transmitida ao destino representado por host juntamente com o recurso.
- *coap_client* –m get [host][/recurso], onde as únicas diferenças em relação à requisição anterior são o nome do método e a omissão da opção –e, já que nenhum *payload* vai ser transmitido neste caso.

Enquanto os dados são transmitidos na interface de rede local da máquina, os pacotes são capturados pelo programa Wireshark. Nos testes foi utilizada a versão 1.12.1 do software.

6. DISCUSSÃO DOS RESULTADOS

Neste capítulo são expostos os resultados dos testes descritos na seção 5.5. Com bases nestes resultados é elaborada uma discussão abordando, sobretudo, a conformidade entre as características descritas para cada protocolo no capítulo 4 e o funcionamento efetivo dos mesmos durante os testes realizados. Toda a discussão baseia-se nas informações colhidas com o auxilio do Wireshark através da captura dos pacotes em tráfego na rede.

6.1. Resultados com MQTT

A transmissão das mensagens funcionou conforme esperado. A aplicação codificada em python utilizando a biblioteca paho-mqtt-client conseguiu realizar a publicação das mensagens da mesma forma que o *mosquitto* conseguiu recuperar as informações através da subscrição ao tópico **temperatura**. A Figura 15 mostra a lista de pacotes capturada no momento em que o assinante faz a subscrição do tópico. É possível verificar que anteriormente à requisição de subscrição outros dois pacotes foram capturados: um comando de conexão e, em seguida, um *ACK* da conexão. Posterior à requisição de subscrição há também o *ACK* da subscrição.

Figura 15: Pacotes capturados no processo de subscrição a um tópico.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000143000	127.0.0.1	127.0.0.1	MQTT	105	Connect Command
6	0.000288000	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
8	0.000520000	127.0.0.1	127.0.0.1	MQTT	86	Subscribe Request
9	0.000599000	127.0.0.1	127.0.0.1	MQTT	71	Subscribe Ack

Fonte: Próprio autor.

As Figuras 16 e 17 mostram, respectivamente, os *payloads* dos pacotes de *CONNECT* (frame nº 4) e *SUBSCRIBE* (frame nº 8) que foram capturados no processo de subscrição do tópico. Na Figura 17 é possível ver os campos do cabeçalho da mensagem que compõem o cabeçalho variável de uma mensagem do tipo *CONNECT*, descritos na seção 4.1.1, como por exemplo, as *flags* para usuário e senha, para fins de autenticação. Na figura 17 é mostrada a pilha de protocolos que foi identificada na captura, seguida das especificações do MQTT contidas no pacote. Pode-se ver que o tópico subscrito e o nível de QoS são partes do *payload*, conforme citado na seção 4.1.1.

Figura 16: Payload de requisição *CONNECT*

Fonte: Próprio autor.

Figura 17: Payload da requisição *SUBSCRIBE*.

Fonte: Próprio autor.

A Figura 18 mostra a tela do terminal de comandos do Linux no momento em que os dados foram recebidos pelo assinante do tópico. Reiterando que as mensagens chegam juntas, pois o caractere # foi usado na subscrição para incluir todos os subtópicos de **temperatura**.

Figura 18: Mensagens recebidas pelo assinante do tópico temperatura

```
[ismael@regulus paho-mqtt-1.0]$ mosquitto_sub -t temperatura/#
32 oC
305 oK
89.6 oF
```

Fonte: Próprio autor.

A lista de pacotes capturados pelo Wireshark quando da execução da aplicação em python está ilustrada na Figura 19. Nela é importante destacar uma coisa: à primeira vista, o pacote nº 9 (*Disconnect Req*) pode parecer um tanto deslocado na sequência, uma vez que mais duas publicações são feitas em seguida, mas há uma explicação para ele estar ali.

Figura 19: Pacotes capturados no processo de publicação das mensagens ao *broker*

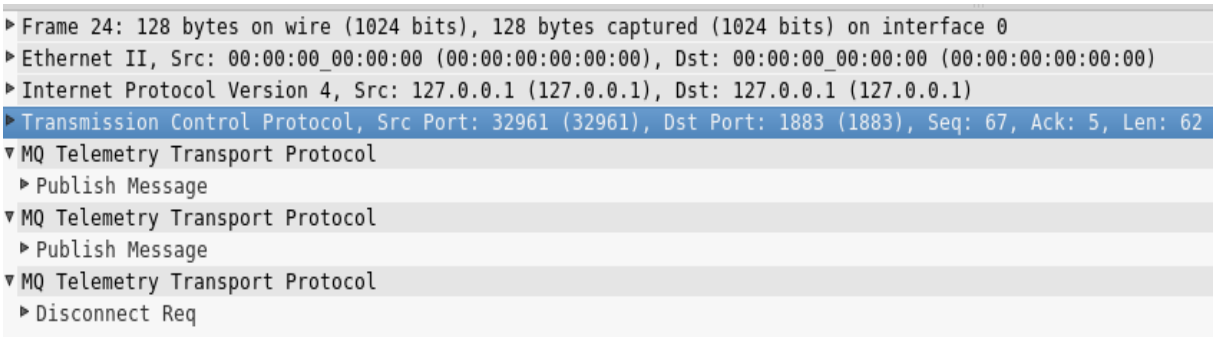
No.	Time	Source	Destination	Protocol	Length	Info
9	0.001009000	127.0.0.1	127.0.0.1	MQTT	103	Connect Command
11	0.001115000	127.0.0.1	127.0.0.1	MQTT	70	Connect Ack
15	0.001367000	127.0.0.1	127.0.0.1	MQTT	93	Publish Message
16	0.001415000	127.0.0.1	127.0.0.1	MQTT	93	Publish Message
24	0.001825000	127.0.0.1	127.0.0.1	MQTT	129	Disconnect Req
25	0.001903000	127.0.0.1	127.0.0.1	MQTT	94	Publish Message
28	0.002004000	127.0.0.1	127.0.0.1	MQTT	99	Publish Message

Fonte: Próprio autor.

Conforme explicado na seção 5.5, a aplicação faz a publicação com um método que transmite várias mensagens de uma só vez, ou seja, a conexão é aberta, todas as mensagens são transmitidas ao *broker* e depois a conexão entre o cliente/*broker* é desfeita. Estes dois pacotes abaixo da requisição *DISCONNECT* são publicações que o *broker* faz ao assinante.

Após receber a mensagem, o *broker* a encaminha a cada um dos assinantes que se conectaram a ele durante a requisição *SUBSCRIBE*. As publicações ao *broker* acontecem nos pacotes nº 15 e 24, enquanto as publicações que ele faz ao assinante acontecem nos pacotes 16, 25 e 26. A Figura 20 mostra que no pacote nº 24 contém, além da requisição de *DISCONNECT*, outras duas publicações feitas de uma vez só para o *broker*.

Figura 20: Requisições *PUBLISH* e *DISCONNECT* no mesmo pacote.



Fonte: Próprio autor.

6.2. Resultados com o CoAP

A aplicação que utiliza o CoAP também mostrou-se capaz de transmitir as mensagens para o servidor, bem como alterar o recurso criado no servidor e também recuperar os atributos contidos neste recurso. Por mais que foram feitas adaptações na aplicação servidora, com base na codificação da biblioteca *libcoap* para adequar-se aos testes definidos na seção 5.3, a aplicação cliente (o código fonte desta não foi alterado) foi capaz de executar as requisições. A Figura 21 mostra a lista de pacotes capturados em todo o teste, ou seja, requisição *GET* com o valor inalterado (sem especificar escala telemétrica), a requisição *PUT* com o novo valor para a temperatura, e outras duas requisições *GET* (Kelvin e Fahrenheit) com o novo valor.

Figura 21: Pacotes capturados nos testes com a aplicação CoAP

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	::1	::1	CoAP	78	CON, MID:34038, GET, /temperature
2	0.000171000	::1	::1	CoAP	72	ACK, MID:34038, 2.05 Content (text/plain)
3	154.3099470(	::1	::1	CoAP	81	CON, MID:5755, PUT, /temperature (text/plain)
4	154.3100950(	::1	::1	CoAP	66	ACK, MID:5755, 2.05 Content
5	195.9965490(	::1	::1	CoAP	80	CON, MID:18313, GET, /temperaturek?
6	195.9967230(	::1	::1	CoAP	73	ACK, MID:18313, 2.05 Content (text/plain)
7	205.8945810(	::1	::1	CoAP	80	CON, MID:8036, GET, /temperaturef?
8	205.8947110(	::1	::1	CoAP	75	ACK, MID:8036, 2.05 Content (text/plain)

Fonte: Próprio autor.

A Figura 22 mostra o terminal de comandos com a sintaxe da requisição *GET* e logo na linha abaixo é exibido o cabeçalho da mensagem, e em seguida informação representando

a temperatura. Os detalhes do payload da requisição podem ser vistos na Figura 23, que traz o pacote *ACK* referente ao *GET*, onde é possível ver o dado no formato *text/plain*[7].

Figura 22: Requisição *GET* feita via cliente CoAP ao servidor

```
[ismael@regulus examples]$ ./coap-client -m get coap://[::1]/temperature
v:1 t:0 tkl:0 c:1 id:34038
22°C
[ismael@regulus examples]$
```

Fonte: Próprio autor.

Figura 23: Payload do pacote *ACK* referente a requisição *GET*.

```
▼ Constrained Application Protocol, Acknowledgement, 2.05 Content, MID:34038
    01.. .... = Version: 1
    ..10 .... = Type: Acknowledgement (2)
    .... 0000 = Token Length: 0
    Code: 2.05 Content (69)
    Message ID: 34038
    End of options marker: 255
  ▼ Payload: Payload Content-Format: text/plain; charset=utf-8 (no Content-Format), Length:
      Payload Desc: text/plain; charset=utf-8
    ▼ Line-based text data: text/plain
        22\302\260C
```

Fonte: Próprio autor.

Como citado anteriormente na seção 5.5, a sintaxe da requisição *PUT* não tem muita diferença em relação à *GET* a não ser pela adição do *payload* da mensagem. A Figura 24 mostra a linha onde foi feita a requisição com o novo valor para a temperatura (35) no terminal de comandos do Linux. Dessa vez apenas o cabeçalho é retornado.

Figura 24: Requisição *PUT* feita via cliente CoAP ao servidor

```
[ismael@regulus examples]$ ./coap-client -m put -e 35 coap://[::1]/temperature
v:1 t:0 tkl:0 c:3 id:5755
[ismael@regulus examples]$
```

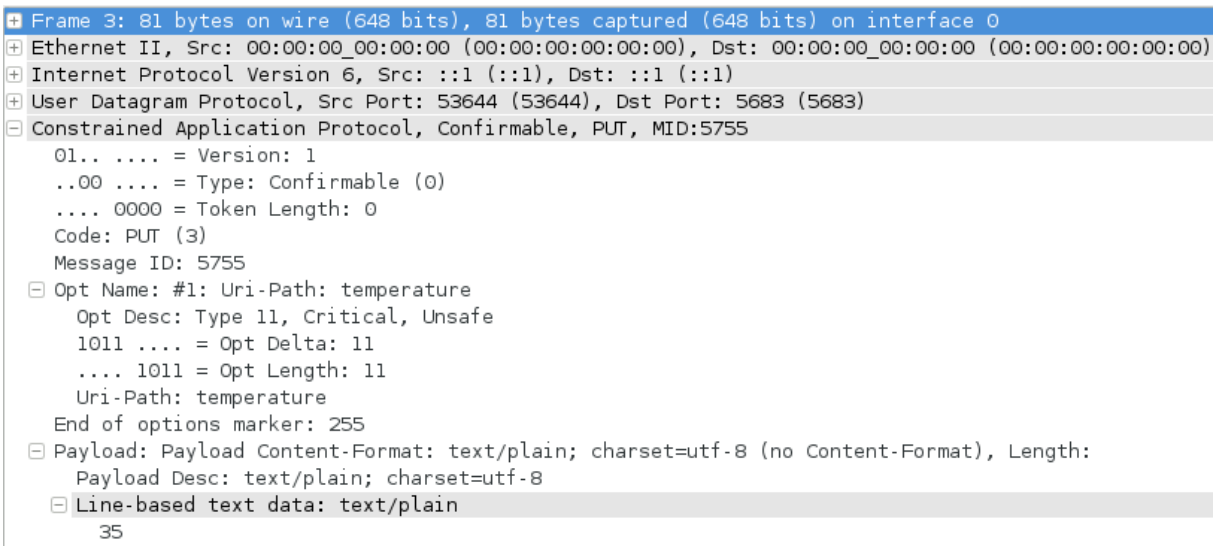
Fonte: Próprio autor.

Os detalhes do pacote da requisição *PUT* capturado pelo Wireshark estão na Figura 25, onde é possível ver, inclusive, o conteúdo que esta sendo enviado ao recurso no servidor.

[7] A sequência de caracteres \302\260 é a representação Unicode do símbolo º. Ela é utilizada porque o terminal não imprima o símbolo corretamente em razão da codificação de caracteres.

Nota-se que, da mesma forma que na requisição *GET*, os mesmos dados do cabeçalho contidos na captura são aqueles retornados pela aplicação no terminal de comandos.

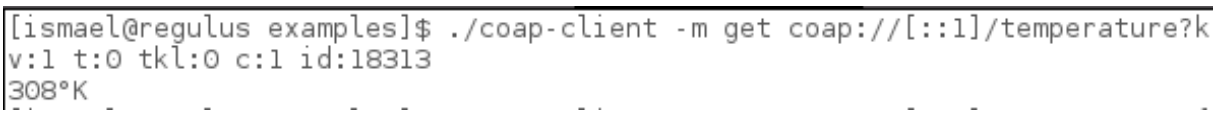
Figura 25: Payload do pacote da requisição *PUT*



Fonte: Próprio autor.

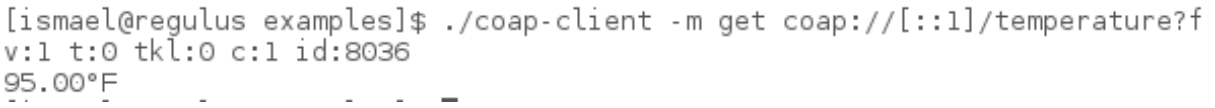
Para recuperar a representação da temperatura nas escalas Kelvin e Fahrenheit apenas adiciona-se a *query string* '?k' ou '?f' ao destino da requisição. As Figuras 26 e 27 mostram o para requisições *GET* nas escalas Kelvin e Fahrenheit, respectivamente, no terminal de comandos do Linux.

Figura 26: Requisição *GET* feita para escala Kelvin



Fonte: Próprio autor.

Figura 27: Requisição *GET* feita para escala Fahrenheit



Fonte: Próprio autor.

7. CONSIDERAÇÕES FINAIS

Através dos testes realizados constatou-se que ambas as implementações dos protocolos estudados cumprem as características descritas neste trabalho com base nas suas respectivas documentações. Entretanto, chegou-se a conclusão de que apenas uma análise do funcionamento dos protocolos em aplicações que executam sobre um ambiente controlado não é suficiente para determinar qual dos dois é a melhor opção para uma aplicação M2M. Neste sentido, indica-se que, para resultados mais satisfatórios, seja desenvolvido um ambiente real, composto de nós sem fio que realmente tenham capacidades limitadas de conexão, processamento e memória capazes de trabalhar com os protocolos estudados, além de não somente analisar o funcionamento do ponto de vista da transmissão das mensagens, mas também o desempenho dos protocolos enquanto os dados estão sendo transportados neste ambiente, desde seu envio, enquanto eles estão sendo transportados no meio físico, até o recebimento no destino.

O cenário M2M abre, sem dúvidas, um vasto campo de possibilidades a serem exploradas pelo profissional da área de análise e desenvolvimento de sistemas. Durante a pesquisa e desenvolvimento que tiveram como resultado esta monografia, várias questões foram levantadas, mas por não pertencerem ao escopo definido, não são discutidas aqui, como por exemplo: quais os meios de conexão disponíveis para aplicações M2M? Quais seriam os padrões de desempenho para aplicações M2M? Como garantir a segurança das informações transmitidas? Ou ainda questões ligadas às aplicações propriamente ditas, como quais seriam os requisitos e qualidade esperada as aplicações M2M? Todas estas questões abrem precedentes para trabalhos futuros concernentes à *Machine-to-Machine* e/ou Internet das coisas, por isso é necessário incentivar o desenvolvimento pesquisas neste cenário, aproximando cada vez mais a sociedade da realidade da computação ubíqua.

REFERÊNCIAS

AGGARWAL, Charu C. The **Internet of Things**: a survey from the data-centric perspective. In: Managing and Mining Sensor Data. Springer, 2013. p. 383 – 428. (Capítulo 12).

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. **The Internet of things**: a survey. The **International Journal of Computer and Telecommunications Networking**, Vol. 54, Issue 15, Elsevier, 28 out. 2010.
Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128610001568>. Acesso em: 1 maio 2014 às 22h30min.

ATZORI, Luigi. et al**. The Social internet of things (SIoT)** when social networks meet the internet of things: concept, architeture and network characterization. **The International Journal of Computer and Telecommunications Networking**, Volume 54, Issue 15, Elsevier, 14 nov. 2012.
Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128612002654>. Acesso em: 1 maio 2014 às 22h50min.

BANDYOPADHYAY, Soma. et al. **Role of middleware for internet of things**: a study. **International Journal of Computer Science and Engineering Survey**, Vol. 2 Nº 3, Tamil Nadu/India, Ago 2011.
Disponível em: <http://airccse.org/journal/ijcses/papers/0811cses07.pdf>. Acesso em: 1 maio 2014 às 22h05min (Publicado por AIRCC).

BERGMANN, Olaf. **Libcoap**: C-implementation of CoAP. 2010. Disponível em: <http://sourceforge.net/projects/libcoap>. Acesso em 1 jul. 2014.

BOSWARTHICK, David; ELLOUMI, Omar; HERSENT, Olivier.
M2M COMMUNICATIONS: A Systems approach. West Sussex/UK: John Wiley & Sons Ltd, 2012. 332p.

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE.**TS 102690 version 2.1.1:** Especificação técnica. Out. 2013. 332 p. Disponível em: <http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/02.01.01_60/ts_102690v020101p.pdf>. Acesso em 9 set. 2014.

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE. **Machine-to-Machine Communications Activity Report 2013**. Disponível em:

<http://portal.etsi.org/TBSiteMap/SmartM2M/ActivityReport.aspx>. Acesso em 1 set. 2014.

EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE. **Overview of ETSI TC M2M Activities**. 2012. Disponível em: <http://docbox.etsi.org/smartm2m/open/information/m2m_presentation.pdf>. Acesso em 9 set. 2014.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures.** Dissertação (PhD em Ciência da Computação e Informação) – Universidade da California, Irvine, 2000. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Acesso em 18 set. 2014.

GUBBI, Jayavardhana; et al. **Internet of Things (IoT):** A vision, architectural elements. And future directions. The International Journal of Grid Computing and Science. Vol. 29, Issue 7, Elsevier, set. 2013. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X13000241>. Acesso em: 9 jun. 2014 às 23h50min.

INTERNACIONAL BUSINESS MACHINES CORPORATION; EUROTECH. MQTT **Specification version 3.1**. out. 2010. Disponível em: <public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html>. Acesso em 26 ago. 2014.

INTERNET ENGINEERING TASK FORCE. **Constrained RESTful Environments:** Charter core. 2010. Disponível em: <http://datatracker.ietf.org/doc/charter-ietf-core/>. Acesso em 6 jun. 2014.

INTERNET ENGINEERING TASK FORCE. **Constrained Application Protocol**: Request for Comments 7252. Jun. 2014. Disponível em: <https://tools.ietf.org/html/rfc7252>. Acesso em 26 ago. 2014.

JAFFEY, Toby. **MQTT and CoAP, IoT Protocols**. 2014. Disponível em: <http://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php>. Acesso em 16 set. 2014.

MASUD, Mehedi; SAID, Omar. **Towards internet of things**: survey and future vision. **International Journal of Computer Networks**, Vol. 5, issue 1, Kuala Lumpur/Malásia, 22 fev. 2013. Disponível em: <http://www.cscjournals.org/csc/manuscript/Journals/IJCN/>. Acesso em 9 jun. 2014.

MESSAGE QUEUE TELEMETRY TRANSPORT. Disponível em: <http://mqtt.org>. Acesso em 15 set. 2014.

MIAO, Wu. et. al. **Research on the architecture of Internet of things**. In.: 3rd IEEE International Conference on Advanced Computer Theory and Engineering (ICACTE). Sichuan, China, 21 ago. 2010. p. 484-487. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5579493&searchWithin%3Dp_Authors%3A.QT.Miao+Wu.QT.%26searchWithin%3Dp_Author_Ids%3A38194913100>. Acesso em 21 ago. 2014.

PAHO PROJECT. Disponível em: <http://eclipse.org/paho>. Acesso em 1 jul. 2014.

PIPER, Andy. **MQTT Wiki.** 2 dez. 2013. Disponível em: <https://github.com/mqtt/mqtt.github.io/wiki/>. Acesso em 15 set. 2014.

POLSONETTI, Chantal. Understand the difference between IoT and M2M. **Chemical Processing** . 24 abr. 2014. Disponivel em: <http://www.chemicalprocessing.com/articles/2014/understand-the-difference-between-iot-and-m2m>. Acesso em 15 set. 2014.

SHELBY, Zach. **CoAP: The Web of Things Protocol.** 30 ago. 2014. Disponível em: <http://pt.slideshare.net/zdshelby/coap-tutorial>. Acesso em 16 set. 2014.

WEISER, Mark. The Computer for the 21st Century. **Scientific Amercian**, Scientific American Magazine, p. 94 – 104, 1 set. 1991.
Disponível em <http://wiki.daimi.au.dk/pca/_files/weiser-orig.pdf>. Acesso em 9 jun. 2014 às 22h30min.

XIAOCONG, Qian; JIDONG, Zhang. **Study on the Structure of "Internet of Things (IOT)" Business Operation Support Platform.** In.: 12th IEEE International Conference on Communication Technology (ICCT). Nanjing, China, 11 nov. 2010. p. 1068 – 1071.
Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5688537&queryText%3DStudy+on+the+structure+of+internet+of+things>. Acesso em 1 nov. 2014.