FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Segurança da Informação

Marcos Flávio Eli Pereira

ANÁLISE DO PROTOCOLO MQTT EM UM DISPOSITIVO DE RECURSOS LIMITADOS

Americana, SP

2018

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Segurança da Informação

Marcos Flávio Eli Pereira

ANÁLISE DO PROTOCOLO MQTT EM UM DISPOSITIVO DE RECURSOS LIMITADOS

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Segurança da Informação, sob a orientação do professor do Prof. Mestre Rossano Pablo Pinto.

Área de concentração: Segurança de Informação

Americana, SP.

2018

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

P493a PEREIRA, Marcos Flavio Eli

Análise do protocolo MQTT em um dispositivo de recursos limitados. / Marcos Flavio Eli Pereira. – Americana, 2018.

69f.

Monografia (Curso de Tecnologia em Segurança da Informação) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Rossano Pablo Pinto

1 Internet das coisas 2. Arduino 3. Criptografia I. PINTO, Rossano Pablo II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.518

681.31

Marcos Flávio Eli Pereira

Análise do Protocolo MQTT em um dispositivo de recursos limitados

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Segurança da Informação pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Segurança da Informação

Americana, 13 de Agosto de 2018.

Banca Examinadora:

Prof. Rossano Pablo Pinto
Mestre
Fatec Americana

Prof. Wladimir da Costa
Mestre
Fatec Americana

Prof. Alberto Martins Junior
Mestre
Fatec Americana

AGRADECIMENTOS

Agradeço a Faculdade Técnica de Americana e a todo o seu corpo docente que faz desta instituição um local sério e colaborativo para o desenvolvimento do ser humano. Em especial agradeço ao Prof. Mestre Rossano que primeiramente acreditou no meu trabalho e orientou-me com disposição e atenção.

DEDICATÓRIA

Ao iniciarmos uma trilha temos o pensamento comum de que o objetivo maior só é alcançado ao final do trajeto. Caminhamos por diversos lugares incríveis que muitas vezes passam desapercebidos. Dedico este trabalho a minha esposa Luciana que caminhou ao meu lado, deu todo o seu apoio e me ajudou a perceber que o verdadeiro valor de uma jornada está na capacidade de superação e aprendizado perante os obstáculos que surgem no caminho. Me transformei em uma pessoa mais forte, apreciei todo o trajeto, venci as dificuldades e o objetivo final tornou-se um agradável detalhe.

RESUMO

Este trabalho põe a prova a capacidade computacional de um dispositivo de recursos limitados desempenhando a função de um sensor. Os sensores são comumente encontrados em infraestruturas de redes para Internet das Coisas. As análises dos protocolos escolhidos demonstram o desempenho e segurança do sensor, durante a transferência de dados em duas situações: a primeira é utilizando o protocolo *MQTT* em sua configuração padrão; e a segunda é em relação ao desempenho do mesmo sensor com os requisitos de Segurança da Informação aplicados.

Palavras Chave: Internet das Coisas; Arduino; Criptografia; Comunicação de dados

ABSTRACT

This work tests the computational capacity of a limited resource device performing the function of a sensor. Sensors are commonly found in Internet networks infrastructures of Things. The analysis of selected protocols demonstrate the performance and security sensor during data transfer in two situations: first, using the protocol MQTT in its standard configuration; and the second is in relation to the performance of the same sensor with the Information Security requirements applied.

Keywords: *Internet of things; Arduino; Encryption; Data communication*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Com o avanço da tecnologia e a miniaturização dos computadores surgiram dispositivos pequenos com recursos otimizados que são capazes de coletar, processar e transmitir dados à rede. O aspecto de baixo consumo energético e a versatilidade de aplicação destes dispositivos contribuíram com o desenvolvimento prático e abrangente do conceito Internet das Coisas, do Inglês *Internet of Things*, comumente referenciado por sua sigla *IoT*.

Um mundo onde todas as coisas são interconectadas por redes de comunicação e interagem entre si, seja por interação humana ou de forma autônoma entre dispositivos é uma das premissas de *IoT*.

Um sensor é um dispositivo com a característica de ler uma variável do ambiente, converter o valor em sua representação digital e transmitir para outro dispositivo, sensores são os elementos mais comuns de um ambiente *IoT.* Geralmente são utilizados vários sensores interconectados em rede para fornecerem dados a uma determinada aplicação, como por exemplo, detectar o número de carros que passam por determinados pontos de uma rodovia e disponibilizar estes dados a um software que mapeia as condições do trânsito de forma autônoma. Como resultado, o software gera ações de otimização do trânsito. Assim, surge um novo campo de percepção onde podemos identificar condições de um ambiente e se antecipar a determinados eventos indesejados, considerando os sensores como uma extensão da percepção humana sobre todas as coisas.

Implementar e manter uma infraestrutura *IoT* requer os mesmos cuidados do que qualquer infraestrutura de redes de informática somada a alguns fatores complicantes. Um dos fatores é o problema de atualização de *firmware* relacionado aos sensores. Muitos dispositivos não recebem as atualizações de segurança necessárias ou muitas vezes não são atualizados pelos fabricantes, o que tornam os sensores vulneráveis, muitos permanecem com a senha de acesso de configuração padrão permitindo a fácil obtenção de acesso. Eles também são alvos cobiçados de ataques com foco específico na criação de redes de “dispositivos zumbis”, denominadas de *botnets*, das quais são utilizadas pelo atacante para realizar ataques a infraestrutura de redes de terceiros, um dos métodos mais conhecidos é o ataque *DDoS (Distributed Denial of Service)* que causa a negação do serviço fornecido pela vítima por consequência ao número excessivo de requisições de

conexão. Outro problema comum é manter o sigilo e proteção dos dados transmitidos. Como os sensores são equipamentos com recursos limitados, os métodos atuais de criptografia devem ser adequados a estes equipamentos e isto requer recurso computacional compatível. Na transmissão de dados pode ocorrer a captura de informações pessoais ocasionando a exposição do usuário. Considere uma “casa inteligente” onde em uma rede *IoT* trafegam imagens, voz e mensagens de controle de diversos equipamentos. O vazamento de informações ocasiona um prejuízo ao usuário que pode não ser mensurável. Outra preocupação é a utilização de *IoT* na área de saúde que hoje já conta com diversos dispositivos de uso pessoal e está em rápida expansão.

A criptografia possibilita a criação dos mecanismos de autenticação que confirma e atesta a identidade de um sensor e possibilita o sigilo das informações durante a transmissão de dados. A junção de um sensor, protocolo de comunicação e criptografia é um desafio para a implementação de uma rede *IoT.* Este trabalho implementa uma rede, testa a eficácia do protocolo escolhido e relaciona as ações implementadas aos mecanismos de controles de SI propostos pela Norma ISO 27002.

É neste cenário, onde existem inúmeros sensores coletando e transmitindo dados, que surgem os novos desafios para garantir a segurança da informação segundo os aspectos de confidencialidade, integridade e disponibilidade dos dados em dispositivos com recursos de processamento limitados.

1.1 MOTIVAÇÃO

O uso de sensores e suas aplicações em rede vem crescendo a cada dia e cada vez mais utilizamos os serviços prestados por eles, muitas vezes de forma transparente e sem percebemos o impacto desta tecnologia ao nosso redor. Garantir o aspecto de segurança da informação protegendo os dispositivos de hardware, os dados gerados, a comunicação de rede e por consequência os novos serviços prestados é a motivação deste trabalho.

1.2 OBJETIVOS GERAIS

Avaliar o impacto dos métodos de segurança da informação em dispositivos

com recursos computacionais limitados que são comumente utilizados em *IoT*.

1.3 OBJETIVOS ESPECÍFICOS

Comparar o desempenho e segurança de um sensor, na transferência de dados, utilizando o protocolo *MQTT* em sua forma padrão em relação ao desempenho do mesmo dispositivo com os requisitos de SI aplicados.

1.4 DELIMITAÇÃO DO TEMA

Utilizar um ambiente de testes composto por dispositivos reais, com recursos limitados, interconectados por uma rede *WLAN* utilizando a pilha de protocolos *TCP/IP e* sob esta pilha utilizaremos o protocolo *MQTT*.

1.5 ESTRUTURA DO TRABALHO DE CONCLUSÃO DE CURSO

O Capítulo 1 aborda uma visão contemporânea do termo Internet das Coisas e o uso de sensores, aborda em linhas gerais os assuntos tratados ao decorrer deste trabalho bem como a motivação de realizá-lo, trata os objetivos gerais e específicos e delimita o tema. O Capítulo 2 traz a bagagem teórica para os assuntos tratados, buscando elucidar as questões futuras. O Capítulo 3 aborda o protocolo *MQTT* e os seus detalhes de projeto e funcionamento. Já o Capítulo 4 traz o protótipo montado e sob esta infraestrutura propõe dois cenários para geração de amostras de pacotes *MQTT*, captura, observação e análise. E finalmente o Capítulo 5 faz a conclusão levando em consideração todo o desenvolvimento dos capítulos anteriores.

2 CONCEITOS

2.1 INTERNET DAS COISAS (IOT - INTERNET OF THINGS)

A Internet das Coisas (IoT) é um novo paradigma que está rapidamente ganhando espaço no cenário da moderna tecnologia de telecomunicações sem fio. A ideia básica deste conceito é a presença dominante em torno de nós de uma variedade de coisas ou objetos - como IDentificação por RadioFreqüência (RFID), tags, sensores, atuadores, telefones celulares e outros dispositivos, dos quais, através de esquemas de endereçamento únicos, são capazes de interagir uns com os outros e cooperar com seus vizinhos para alcançar objetivos comuns. (ATZORI. 2010)

Quando um objeto tem a capacidade descrita acima, mesmo com restrições de capacidade de processamento, tamanho, memória e consumo de energia elétrica, eles são denominados de objetos inteligentes.

Imagine um mundo onde bilhões de objetos podem sentir, comunicar e compartilhar informações, tudo interconectado através de redes IP públicas ou privadas. Esses objetos interconectados coletam, analisam e usam os dados regularmente para fornecer riqueza de dados para análise e inteligência para o planejamento, gerenciamento e tomada de decisões. Este é o mundo da Internet das Coisas (IOT). (IMDA. 2017. pág. 1)

2.2 ARQUITETURA BÁSICA DE COMUNICAÇÃO EM *IOT*

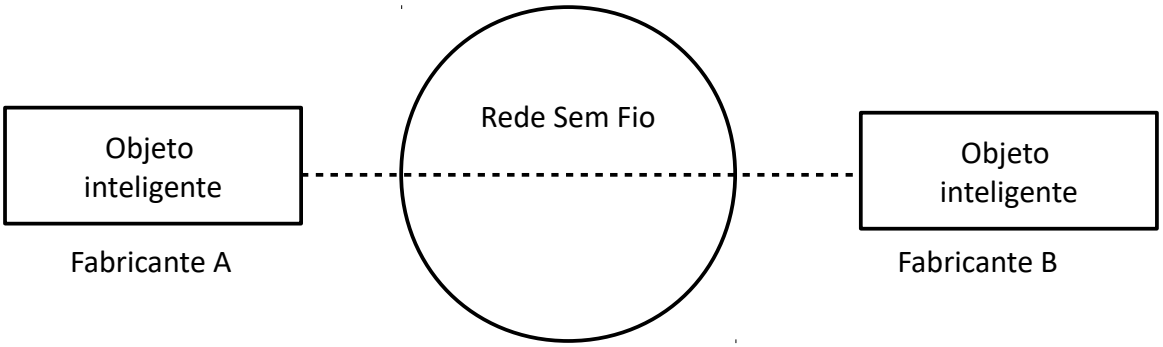
A instituição internacional IETF (Internet Enginnering Task Force), em sua documentação técnica RFC 7452 (Request for Comments), define os padrões de comunicação para objetos inteligentes, conforme, segue:

2.2.1 Device-to-Device

Comunicação direta entre dois objetos inteligentes que foram desenvolvidos por fabricantes diferentes com o propósito de interoperabilidade. Conforme visto na Figura 1. Para que a comunicação seja possível, os fabricantes precisam aderir a uma pilha de protocolos. Esta decisão depende de alguns fatores, como por

exemplo: camada física, suporte IPv6, IPv4 ou ambos por questões de retrocompatibilidade e camada de transporte.

Figura 1 - Padrão de comunicação *Device-to-Device*



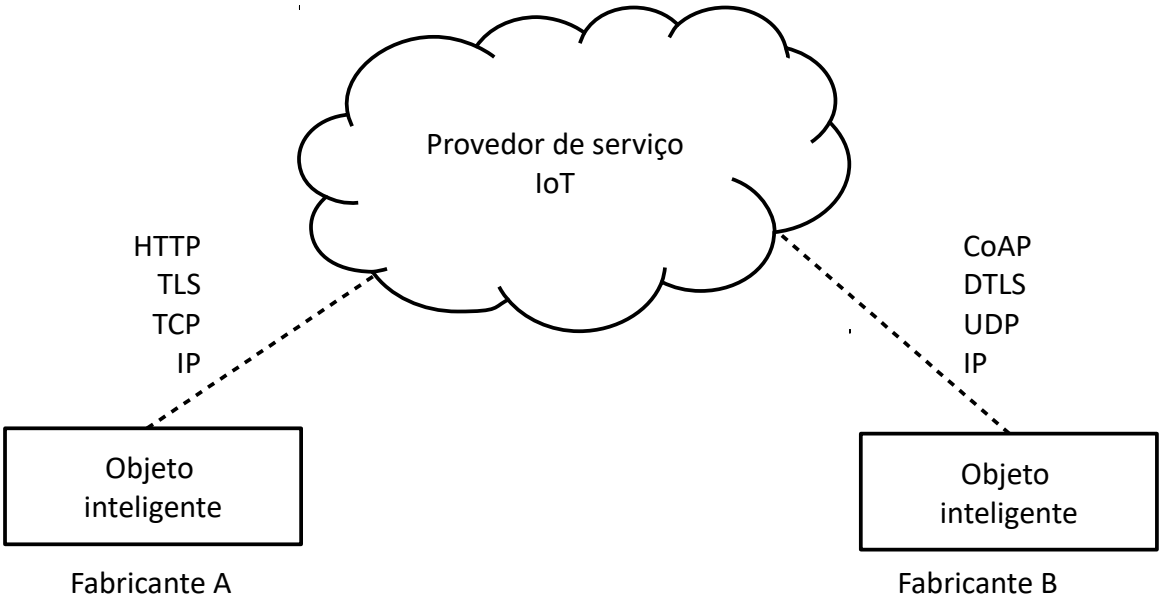
Fonte: RFC 7452, 2015, p.4. Adaptado.

2.2.2 Device-to-Cloud

Neste modelo a comunicação é entre o objeto inteligente e um serviço em nuvem, o dispositivo é capaz de se comunicar via protocolo baseado em IP ponto a ponto, mas existe a dependência de interconexão do objeto com um serviço em nuvem. Vários fabricantes podem utilizar protocolos padrões para permitirem que o seu objeto inteligente se comunique com provedores de serviço *IoT* em nuvem, conforme visto na Figura 2.

Uma desvantagem é a dependência de um serviço em nuvem para a utilização do objeto inteligente.

Figura 2 - Padrão de comunicação *Device-to-Cloud*

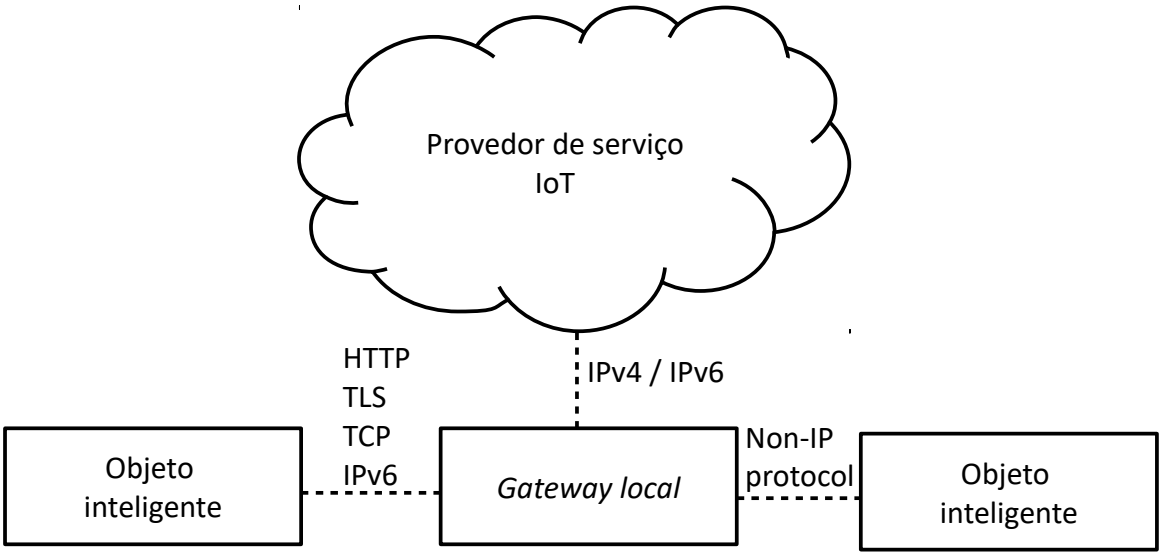


Fonte: RFC 7452, 2015, p.7. Adaptado.

2.2.3 *Device-to-Gateway*

Para casos de interoperabilidade entre tecnologias diferentes ou o uso de dispositivos legados, são introduzidos *gateways* na arquitetura de comunicação criando uma ponte entre tecnologias diferentes e fornecendo outras formas de interconexão em rede e segurança. Veja Figura 3.

Figura 3 - Padrão de comunicação *Device-to-Gateway*



Fonte: RFC 7452, 2015, p.8. Adaptado.

Um exemplo do uso deste padrão de comunicação é o uso do aparelho de telefone celular como *gateway* para permitir a conexão à internet de um relógio de pulso inteligente, que é um dispositivo vestível (*wearables).*

2.2.4 Back-End Data Sharing

Quando são utilizados vários provedores de serviço em nuvem para compor uma aplicação, os objetos enviam dados a um serviço em nuvem, os dados são exportados e enviados para análise em outro provedor de serviço em nuvem, compondo assim uma rede de provedores.

2.3 SENSORES EM *IOT*

A definição de sensor pelo dicionário Michaelis é: “Dispositivo ou equipamento que, sensível a estímulos magnéticos, motores, de calor, de luz, de pressão, de som etc., é capaz de converter essa energia e transmitir um impulso correspondente”.

Dentro do conceito de *IoT*, sensor é um dispositivo de *hardware* com capacidade de ler uma característica do ambiente e transmitir o valor lido, convertido e representado de forma digital por uma rede de comunicação e de forma autônoma.

Um dispositivo que vem se destacando neste cenário é o *System on Chip (SoC)* ESP8266 (ESPRESSIF SYSTEMS, 2017). Suas características são o baixo custo, rede sem fio integrada, baixo consumo energético, portas digitais e analógicas de entrada e saída e a facilidade de se implementar códigos utilizando uma *Integrated Development Environment (IDE)*. Uma placa de projeto aberto que utiliza este *SOC* é conhecida por *NodeMCU* e pode-se utilizar a *IDE* e as bibliotecas já conhecidas do *Arduino* para a sua programação. O *NodeMCU* foi escolhido como objeto de análise deste trabalho por se tratar de um dispositivo com recursos limitados.

2.4 COMUNICAÇÃO E PROTOCOLO

Para que exista a comunicação entre dois pontos é necessário haver uma regra comum para que a mensagem transmitida seja compreendida. De modo análogo a uma conversa entre duas pessoas, onde existem um emissor, um meio de comunicação e um receptor, a comunicação ocorre somente se as duas pessoas compreendem a mesma língua, da qual, podemos considerar como um protocolo de comunicação.

“O termo protocolo é usado para designar um conjunto bem conhecido de regras e formatos a serem usados na comunicação entre processos a fim de realizar uma determinada tarefa” (COLOURIS, 2007, p. 79).

Kurose define da seguinte maneira: “Os protocolos definem o formato e a ordem das mensagens da rede bem como as ações que são tomadas quando da transmissão ou recepção de mensagens” (2012, p. 9).

“Os protocolos permitem que seja observada ou compreendida a comunicação, sem que se conheçam os detalhes de um *hardware* de interligação em redes de um fornecedor específico.” (COMER, 2008, p. 177).

2.4.1 Arquitetura em camadas

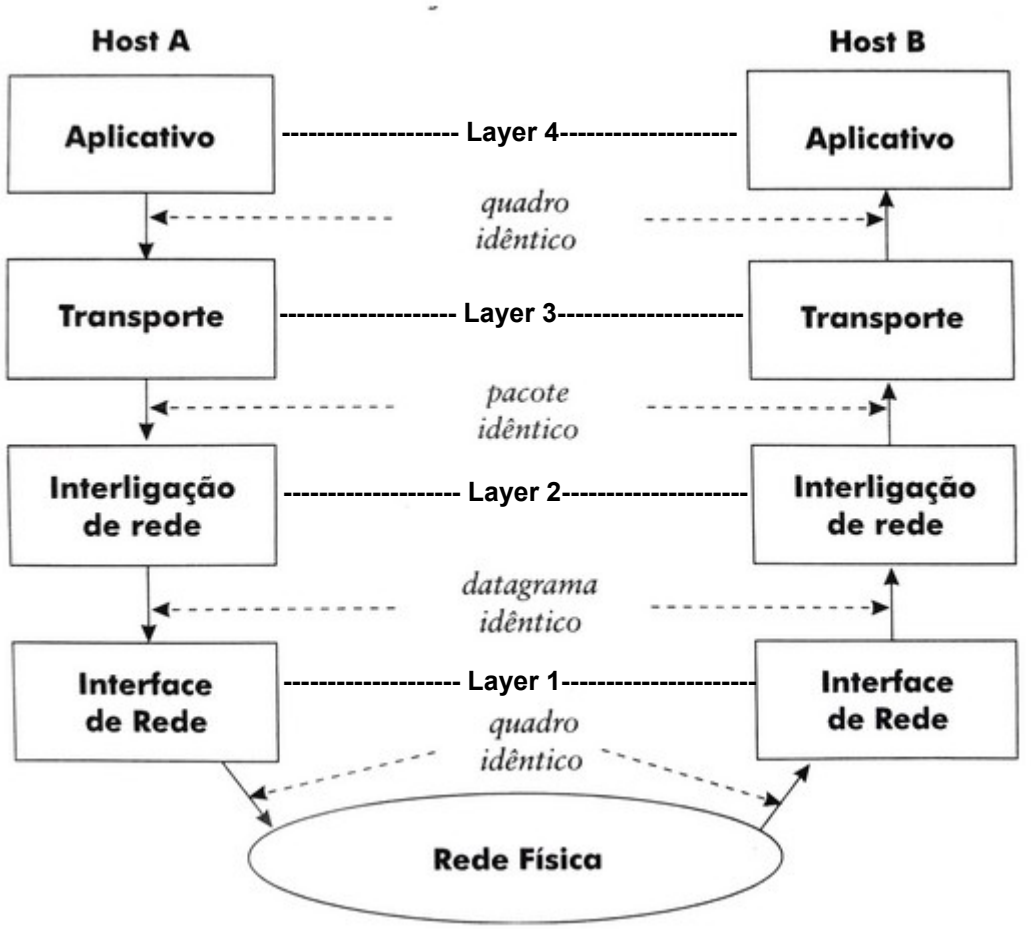
O projeto da maioria das redes é organizado como uma pilha de camadas. Cada camada n é responsável por receber uma mensagem de uma camada adjacente (n+1 ou n-1), processar e repassar o resultado para a próxima camada. Cada camada comunica exatamente com a mesma camada de outro *host* e cada

uma é implementada por um protocolo que é encarregado de interpretar as mensagens e solucionar problemas decorrentes do seu nível na pilha.

A divisão em camadas dos protocolos baseia-se num conceito fundamental denominado de *princípio de divisão em camadas* que resume-se: “Os protocolos divididos em camadas são projetados de modo que a camada n de destino receba exatamente o mesmo objeto enviado pela camada n de origem.” (COMER, 2008, p. 188).

De forma conceitual a Figura 4 apresenta um projeto de rede dividido em 4 camadas *(Layers)* e a interação entre dois clientes de rede (*Host A e Host B)* por um meio físico de interconexão de rede (Rede Física).

Figura 4 - Principio da divisão em camadas.



Fonte: (COMER, 1998, p. 189)

“A camada *n* do *host* B recebe exatamente o mesmo objeto que a camada *n* do *host* A enviou.” (COMER, 1998, p.189)

2.4.2 Modelo OSI (*Open System Inteconection*)

Cada fabricante de *hardware* pode especificar o seu próprio protocolo, mas a tecnologia fica limitada e refém deste, para padronizar e permitir que haja um meio comum e aberto de comunicação de rede entre dispositivos de diversos fabricantes, a *ISO* (*International Organization for Standardization*) especifica o modelo *OSI* (*Open System Inteconection*).

A norma ISO/IEC 7498-1:1994 define que o objetivo do modelo de referência conceitual *OSI* é prover uma base padrão para a coordenação de desenvolvimento com o propósito de interconexão de sistemas.

“O modelo *ISO* é reconhecido como modelo de referência conceitual para a organização em camadas dos protocolos de rede.” (COMER, 1998, p. 181)

Ele é constituído por sete camadas conforme a Tabela 1.

Tabela 1: Camadas do modelo de referência OSI

CAMADA	**FUNCIONALIDADE**	**DESCRIÇÃO**
7	Aplicação	Protocolos de aplicativos, como por exemplo TELNET, FTP, HTTP, MQTT. Esta camada consiste no nível mais alto da pilha que permite a comunicação em alto nível entre aplicativos.
6	Apresentação	Tradução das representações de dados entre o serviço de rede e a aplicação, como por exemplo: codificação de caracteres, compressão de dados e criptografia.
5	Sessão	Provê um mecanismo para iniciar, terminar e gerenciar sessões entre processos de aplicações.
4	Transporte	Oferece confiabilidade ao fazer com que o *host* de destino se comunique com o *host* de origem.
3	Rede	Interação entre o *host* e a rede. Responsável pelo endereço de rede, controle de congestionamento e divisão dos pacotes para transmissão.
2	Enlace de dados	Define o formato dos dados a serem transmitidos, aqui os pacotes são codificados em quadros, representação em bits de como serão transmitidos por sinais elétricos.
1	Física	Interconexão física, incluindo as características elétricas de tensão e corrente.

Fonte: Próprio autor, adaptado de (COMER, 1998)

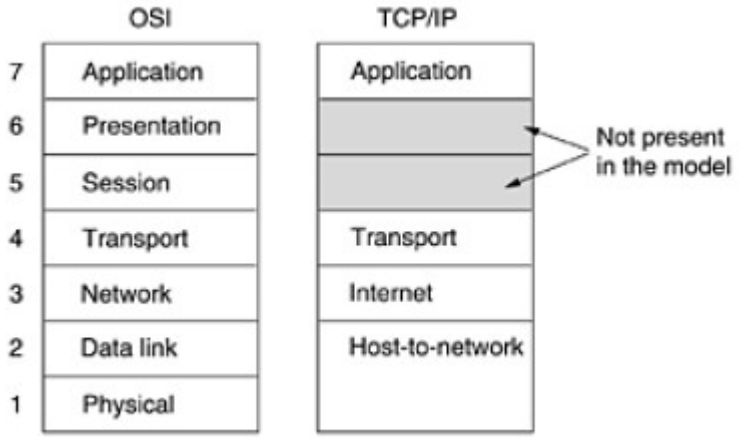
2.4.3 Modelo *TCP/IP*

O modelo *TCP*/*IP* é o utilizado pela *Internet.* Este modelo foi concebido desde o início com o objetivo de conectar várias redes de maneira uniforme.

Kurose descreve o termo *Internet* como uma infraestrutura de redes da qual provê serviços para aplicações distribuídas (2012, p 2).

O modelo *TCP/IP* utiliza somente 4 camadas que podem ser relacionadas ao Modelo de referência *OSI* conforme Figura 5.

Figura 5 - Modelo referência *TCP/IP* e correspondência *OSI*



Fonte: (TANENBAUM, 2003, p. 48)

Na Figura 5 pode-se observar que as camadas 5 e 6 do modelo *OSI* não estão presentes no modelo *TCP/IP* e as funções desempenhadas por elas, normalmente, ficam sob responsabilidade da camada 4 do modelo *TCP/IP*. Percebe-se também que as camadas 1 e 2 do modelo *OSI* são englobadas pela camada 1 no modelo *TCP/IP*.

Convenciona-se o modelo *TCP*/*IP* para a numeração e nomenclatura das camadas tratadas ao decorrer deste trabalho.

2.4.3.1 Camada de Aplicação

O protocolo *MQTT* é um protocolo de aplicação do modelo *TCP*/*IP,* ele contém em seu projeto particularidades das camadas de sessão e apresentação do

modelo *OSI*; o *broker* ao receber uma solicitação de conexão faz a abertura e o gerenciamento da sessão; a camada de apresentação pode ser entendida pelo fato do protocolo padronizar a codificação *UTF-8* para a transferência de dados.

Outro protocolo abordado é o protocolo *TLS* que pode ser considerado como camada de apresentação [REF] do modelo *OSI*, mas não é definida formalmente. Mas o *TLS* também possui elementos da camada de sessão, como o *handshake* para estabelecimento de sessão de comunicação criptografada. Neste *TCC*, o *TLS* será tratado como sendo da camada de aplicação do modelo *TCP/IP*.

A análise de ambos os protocolos *MQTT* e *TLS*, na camada de aplicação do modelo *TCP/IP* é o objetivo deste trabalho. Estes são os protocolos atrelados à *IoT* escolhidos. Por este motivo as demais camadas do modelo *TCP/IP* não serão diretamente abordadas.

Kurose diz que “A camada de aplicação é a razão da existência de uma rede de computadores” (2012, p. 83).

> No nível mais alto, os usuários rodam programas aplicativos que acessam serviços disponíveis através de uma interligação em redes *TCP/IP.* Um aplicativo interage com um dos protocolos do nível de transporte necessário, que tanto pode ser uma sequência de mensagens individuais ou um *stream* contínuo de *bytes.* O programa aplicativo passa, para o nível de transporte, os dados de forma adequada, para que possam, então, ser transmitidos (COMER, 1998, p.185).

Portanto, a camada de aplicação fornece ao desenvolver uma interface de programação, já com a infraestrutura lógica de rede pronta e preparada para transmitir mensagens possibilitando a comunicação entre aplicações. Assim, surgiram os protocolos de aplicação que convencionam o modo da troca de mensagens entre aplicativos com o mesmo objetivo, como por exemplo, o protocolo *HTTP.* Existem vários aplicativos servidores de páginas WEB, bem como, vários aplicativos navegadores, todos padronizados pelo HTTP o que permite o acesso e a troca de informação independente do fornecedor dos aplicativos.

2.5 SEGURANÇA DA INFORMAÇÃO

A capacidade e velocidade de produzir, armazenar e analisar dados dentro de um contexto, transformando-o em informação, gerou um novo patamar de relevância da informação, tornando-a o bem mais valioso e

estratégico para uma organização.

“A informação é o conjunto de dados que, se fornecido sob a forma e tempo adequados, melhora o conhecimento da pessoa que recebe, e a habilita a desenvolver melhor determinada atividade, ou a tomar decisões melhores” (CARVALHO. 2000, p. 237).

Dada tamanha importância e impacto da informação em uma organização faz-se necessário preservá-la. Surge então a área da segurança da informação (SI), que é amplamente difundida e discutida no Mundo. Em especial, no Brasil, temos a norma ISO/IEC 27002 que define o termo segurança da informação como: “preservação da confidencialidade, da integridade e da disponibilidade da informação; adicionalmente, outras propriedades, tais como autenticidade, responsabilidade, não repúdio e confiabilidade, podem também estar envolvidas”. Estas propriedades também são conhecidas por pilares da segurança da informação:

• Confidencialidade – Propriedade de que a informação não será disponibilizada ou divulgada a indivíduos, entidades ou processos que não possuam autorização. [ISO/IEC 13335-1:2004].

• Integridade – Propriedade de proteção à precisão e perfeição da informação e de recursos. [ISO/IEC 13335-1:2004].

• Disponibilidade – Propriedade de ser acessível e utilizável sob demanda por uma entidade autorizada. [ISO/IEC 13335-1:2004]

O uso da criptografia é essencial para a prática da SI, durante a transmissão de dados a confidencialidade é obtida por uso de algoritmos de criptografia codificando o conteúdo das mensagens transmitidas. Também se garante a integridade ao utilizar a criptografia, ela fornece meios para atestar de que o conteúdo de uma mensagem não foi alterado durante o processo de transmissão.

“Criptografia é o estudo do projeto das técnicas para garantir o sigilo e/ou a autenticidade da informação” (STALLINGS, XVI).

Na área da criptografia, um texto que passou pelo processo de criptografia e o seu conteúdo não pode mais ser compreendido por seres humanos é denominado

de texto cifrado e o texto original é denominado de texto claro.

2.6 SEGURANÇA DA INFORMAÇÃO, *IOT* E SENSORES

Nota-se um fator importante na definição de Segurança da Informação que é o tempo adequado. Este tempo está intimamente relacionado à disponibilidade dos objetos inteligentes que formam a base da *IOT*, ter a capacidade de gerar, converter e transmitir um dado de forma segura, no formato e tempo adequados ao propósito à que se destinam é o objetivo da implementação e utilização de ambientes *IOT*.

O número de objetos inteligentes a serem utilizados para uma aplicação IOT depende do projeto, necessidade e planejamento da solução, considerando uma aplicação que requer o uso de sensores que normalmente são utilizados em grandes quantidades cria-se um cenário onde há uma grande disponibilidade de diversos dispositivos.

Supõe-se que existe a possibilidade de aquisição de um sensor com uma falha de segurança. Este sensor, ao ser utilizado e conectado a uma rede de uma organização sem a devida preocupação, torna-se um ponto vulnerável a uma infraestrutura de rede que até então era considerada segura. Pelo aspecto do dispositivo de hardware ser pequeno e de função específica, muitas vezes não são tomadas as devidas precauções, em comparação a outros equipamentos de TI, em sua utilização.

O Relatório Anual de Segurança Cibernética da Cisco de 2018[1] informa que "… os adversários já estão explorando as falhas de segurança em dispositivos *IoT* para obter acesso a sistemas, inclusive os sistemas de controle industrial que oferecem suporte a infraestruturas essenciais…" (CISCO. 2018, p. 31).

A grande disponibilidade e variedade de sensores, projetos de hardware livre e utilização de sistemas operacionais conhecidos, aliados a capacidade de interconexão de redes são fatores determinantes para que sejam alvos de ataques.

Garantir a integridade física de um sensor, a comunicação entre os dispositivos, a veracidade dos dados gerados e a disponibilidade é responsabilidade da SI. Para prover esta segurança são utilizados mecanismos de SI.

---

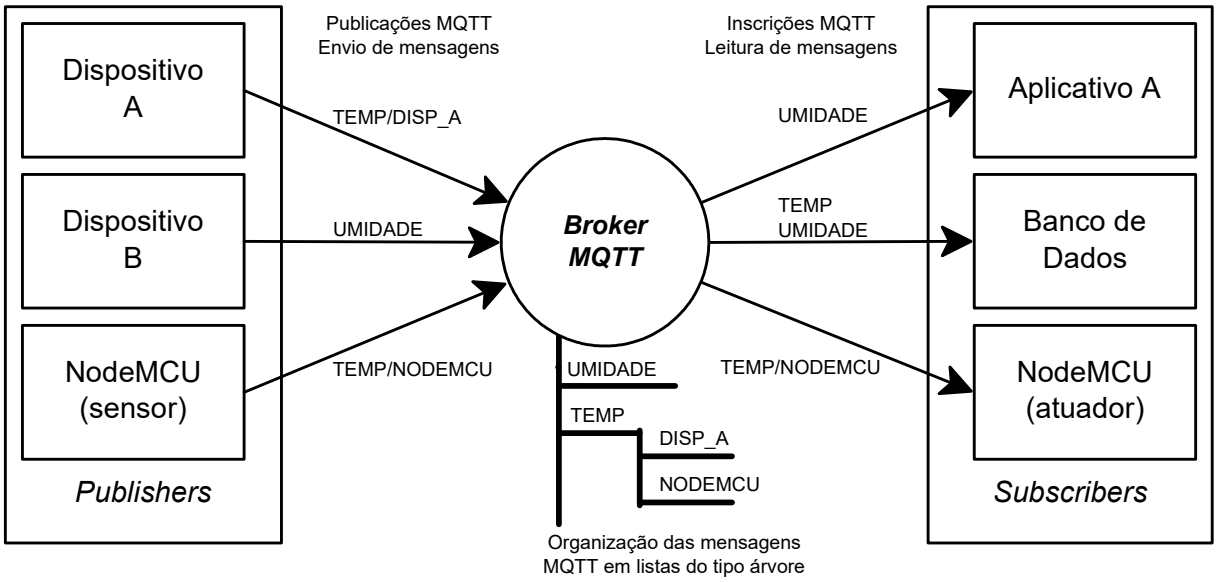
1 A empresa Cisco é a líder mundial em TI e redes, anualmente divulga um documento de acesso público em sua página WEB denominado de Relatório Anual de Cibersegurança. Disponível em: https://www.cisco.com/c/pt_br/products/security/security-reports.html. Acesso em: 10 abr. 2018.

3 PROTOCOLO MQTT

O *MQTT* é um protocolo de transporte de mensagem do tipo publicação/inscrição (*Publish/Subscribe*) cliente e servidor. Projetado com as características de ser leve, aberto, simples e de fácil implementação (OASIS, 2014). O seu uso se torna ideal em situações onde existem restrições de ambiente de rede. Redes *IoT* exigem um protocolo de mensagens com código pequeno e adequado para o uso com pouca largura de banda ou o uso com dispositivos de recursos limitados.

O funcionamento do modelo publicação/inscrição utiliza um programa servidor denominado de *Broker* cuja função é permitir a criação de listas do tipo árvore de diretórios para a organização e distribuição das mensagens. O *Broker* centraliza as mensagens recebidas e as enviam somente para os inscritos nas referentes listas. Existem dois papéis a serem desempenhados pelos clientes *MQTT: Publisher* (editor) que são os clientes que publicam as mensagens em uma ou mais listas; e o *Subscriber* (assinante) que se inscreve em uma ou mais listas e aguarda a publicação de mensagens. Ambos os papéis podem ser desempenhados pelo mesmo dispositivo.

Figura 6 - Funcionamento do protocolo *MQTT*



Fonte: próprio autor

Como descritivo do funcionamento (Figura 6), consideremos o Dispositivo B,

o Aplicativo A e o *Broker*, desta maneira:

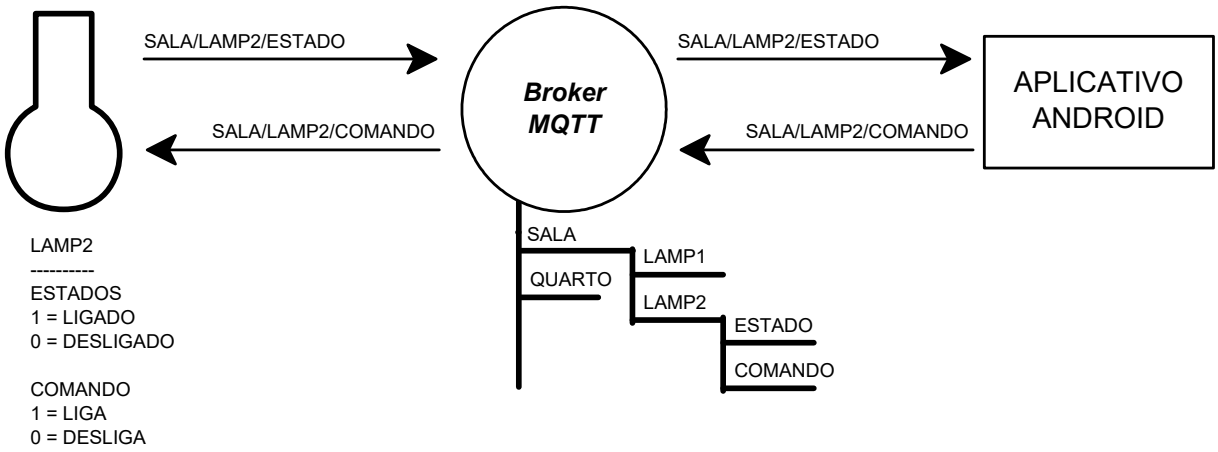
O Aplicativo A conecta-se ao *Broker* e se inscreve na lista UMIDADE, neste momento o Aplicativo A fica na espera da publicação de uma mensagem;

O Dispositivo B conecta-se ao *Broker* e publica uma mensagem contendo um valor na lista UMIDADE;

O *Broker*, ao receber o valor do Dispositivo B, verifica se exitem clientes inscritos na lista e envia a mensagem a cada inscrito, no caso para a Aplicação A, após o envio a mensagem é descartada da lista.

Pode-se ainda exemplificar uma situação onde há a necessidade de dispositivos atuarem em ambos os papéis, *Publisher* e *Subscriber.* Imagine uma “lâmpada inteligente” conectada a uma rede *WiFi* doméstica que pode ser controlada por um aplicativo utilizando o protocolo *MQTT*, conforme Figura 7.

Figura 7 - Cliente *MQTT*: *Publisher* e *Subscriber* ao mesmo tempo



Fonte: próprio autor

Desta maneira é possível enviar comandos à lâmpada e receber a confirmação das ações resultantes do mesmo. Este modelo já serve de base para um sistema autônomo de *IoT.* Imagine uma sala com sensores de luminosidade, lâmpadas e cortinas inteligentes. Basta ajustar a luminosidade desejada para que as cortinas e lâmpadas realizem os ajustes necessários para alcançar tal resultado, podendo até considerar a economia de energia elétrica em relação a variação da luz solar durante o dia.

3.1 ESTRUTURA DO PACOTE DE CONTROLE *MQTT*

A documentação técnica do protocolo *MQTT* refere-se a mensagem como pacote de controle. Esta nomenclatura é utilizada para camadas superiores do modelo *OSI*.

O *MQTT* é um protocolo situado na camada 4, camada de aplicação da pilha de protocolos *TCP/IP. O* seu pacote de controle consiste em três partes: o cabeçalho fixo (*fixed header*), cabeçalho variável (*variable header*) e a carga (*payload*). A menor estrutura do pacote é composta somente pelo cabeçalho fixo. A codificação de caracteres utilizada é o *UTF-8* e qualquer código de caractere transmitido fora deste padrão causa a desconexão do cliente por parte do servidor.

3.1.1 Cabeçalho fixo (*fixed header*)

Todos os pacotes de controle *MQTT* são ser divididos em 3 partes: cabeçalho fixo, cabeçalho variável e carga quando presente, a Tabela 2 em sua primeira coluna define a estrutura do pacote de controle contendo as 3 partes. As colunas restantes definem o formato do cabeçalho fixo, percebe-se que ele é pequeno e contém apenas 2 *bytes*, o primeiro *byte* (*byte* 1 da Tabela 2) é divido em duas partes iguais de 4 *bits* onde os *bits* de 4 à 7 especificam o tipo do pacote (Tabela 3) e os *bits* de 0 à 3 são utilizados como sinalizadores (*flags*) (Tabela 4), todos descritos na seção 3.1.1.1. Se tratando do segundo *byte* (*byte* 2 da Tabela 2) ele contém o comprimento restante do pacote, detalhado na seção 3.1.1.2.

Tabela 2: Formato do cabeçalho fixo

ESTRUTURA
Cabeçalho fixo
Cabeçalho Variável
Carga

FORMATO DO CABEÇALHO FIXO								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
byte 1	Tipo do pacote (*type*)				Sinalizadores (*flags*)			
byte 2...	Comprimento restante *(Remaining Length)*							

Fonte: *MQTT Version 3.1.1* (BANKS; GUPTA. 2014. p. 16)

3.1.1.1 Byte 1, Tipos e Sinalizadores

Cada pacote enviado contém um tipo definido e para cada tipo existem os sinalizadores (*flags*) associados, conforme visto na Tabela 3.

Tabela 3: Tipos dos pacotes *MQTT*

TIPO DO PACOTE	VALOR	DIREÇÃO DO FLUXO	DESCRIÇÃO
Reserved	0		Reservado
CONNECT	1	Cliente para Servidor	Requisição de conexão
CONNACK	2	Servidor para Cliente	Confirmação da conexão
PUBLISH	3	Cliente para Servidor ou Servidor para Cliente	Publicação de mensagem
PUBACK	4	Cliente para Servidor ou Servidor para Cliente	Confirmação da Publicação da mensagem (QoS 1)
PUBREC	5	Cliente para Servidor ou Servidor para Cliente	Publicação recebida, utilizada para entregas garantidas (QoS 2 – parte 2/4)
PUBREL	6	Cliente para Servidor ou Servidor para Cliente	Publicação de resposta à PUBREC (QoS 2 – parte 3/4)
PUBCOMP	7	Cliente para Servidor ou Servidor para Cliente	Publicação de resposta à PUBREL (QoS 2 – parte 4/4)
SUBSCRIBE	8	Cliente para Servidor	Requisição de inscrição em uma lista
SUBACK	9	Servidor para Cliente	Confirmação da requisição de inscrição
UNSUBSCRIBE	10	Cliente para Servidor	Sair de uma lista
UNSUBACK	11	Servidor para Cliente	Confirmação de sair da lista
PINGREQ	12	Cliente para Servidor	PING solcitação
PINGRESP	13	Servidor para Cliente	PING resposta
DISCONNECT	14	Cliente para Servidor	Desconectar do servidor
Reserved	15		Reservado

Fonte: *MQTT Version 3.1.1* (BANKS; GUPTA. 2014. p. 16)

O único tipo de pacote que pode utilizar os sinalizadores é o *PUBLISH* usado na versão 3.1.1 do protocolo, os outros tipos possuem valores fixos, por esta razão, a Tabela 4 somente descreve os sinalizadores do pacote *PUBLISH*.

Tabela 4: Sinalizadores do pacote *PUBLISH*

TIPO DO PACOTE	SINALIZADORES	Bit 3	Bit 2	Bit 1	Bit 0
PUBLISH	MQTT 3.1.1	DUP	QoS	QoS	RETAIN

Fonte: *MQTT Version 3.1.1* (BANKS; GUPTA. 2014. p. 17)

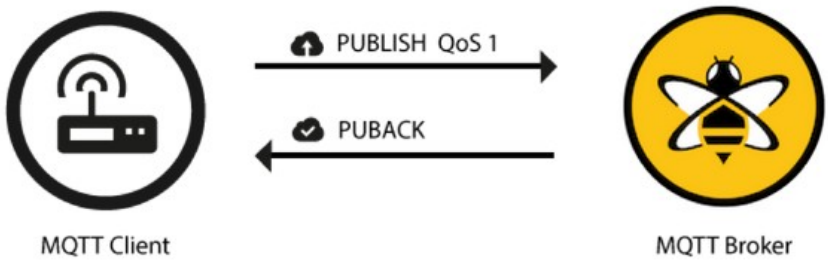
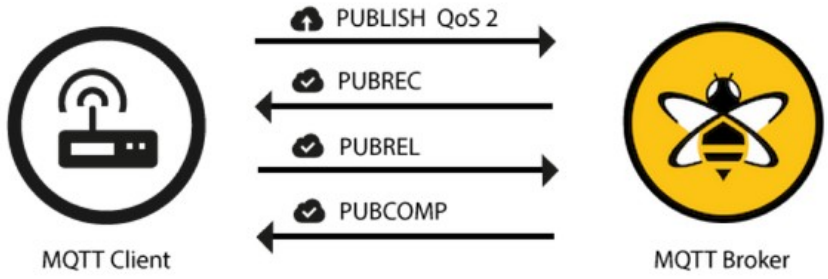
DUP (*bit* 3) = É utilizado quando existe a necessidade do reenvio do pacote, tanto por parte do cliente quanto por parte do servidor;

QoS (*bits* 1 e 2) = Qualidade do Serviço (*Quality of Service*). Estes *bits* definem o nível de garantia de entrega de um pacote, conforme Tabela 5.

Tabela 5: Sinalizadores do pacote *PUBLISH*

VALOR *QoS*	Bit 2	Bit 1	DESCRIÇÃO
0	0	0	Entrega uma única vez, quando não há necessidade da garantia de entrega. Exemplo: Para um sensor que envia o valor de temperatura do ambiente de forma periódica, perder um valor não representa uma falha grave. Figura 8 - Protocolo *MQTT QoS* 0  Fonte: Página HiveMQ MQTT Essentials Part 6[2]

2 Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels. Acesso em: 5 mai. 2018.

1	0	1	Entrega pelo menos uma vez, envia a confirmação de entrega ao receber um pacote. Figura 9 - Protocolo *MQTT QoS* 1  Fonte: Página HiveMQ MQTT Essentials Part 6[2]
2	1	0	Entrega garantida, existe a confirmação da entrega do pacote e a confirmação de que o cliente foi notificado. Figura 10 - Protocolo *MQTT QoS* 2  Fonte: Página HiveMQ MQTT Essentials Part 6[2]

RETAIN (*bit* 0) = Sinalizador de retenção, indica ao servidor para reter a mensagem e enviar a todos que conectarem posteriormente, até que uma nova mensagem com retenção chegue ao servidor.

3.1.1.2 Byte 2, Comprimento restante (Remaining Length)

Este campo de 8 bits, 1 byte, é utilizado para indicar o comprimento total do pacote a partir deste campo. Utiliza uma codificação de 128 valores e um bit de continuação para expressar o valor do comprimento total do pacote (soma do Cabeçalho variável com a Carga), podendo ter no máximo 4 bytes.

Supondo que o tamanho do pacote seja 321 bytes, a representação da codificação 128 deste campo será:

Tabela 6: Comprimento restante (Remaining Length)

		Bits							
byte	**Valor \ Bits**	**bit de continuação 7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	193 (128 + 65)	1	1	0	0	0	0	0	1
2	2	0	0	0	0	0	0	1	0

Fonte: próprio autor.

O primeiro *byte* contém o número 65 (*bits* de 0 à 6) e o *bit* de continuação setado em 1 informando que existe o próximo *byte* contendo o multiplicador. Portanto o número é composto por 65 + 128 * 2, resultando em 321.

3.1.2 Cabeçalho variável

O cabeçalho variável é uma extensão do cabeçalho fixo e o seu conteúdo varia de acordo com o tipo do pacote. Normalmente ele contém um campo de identificação.

Os pacotes de controle que necessariamente devem ter o campo identificador de 16 bits são os tipos *SUBSCRIBE*, *UNSUBSCRIBE* e *PUBLISH* desde que este último utilize o *QoS* maior que zero. Este identificador é muito importante para que haja o controle de confirmação e retransmissão de um pacote e também o gerenciamento da sessão da conexão.

3.1.3 Carga (*Payload*)

O campo carga é utilizado pelos pacotes dos tipos *CONNECT*, *PUBLISH*, *SUBSCRIBE*, *SUBACK* e *UNSUBSCRIBE*.

Somente os pacotes *CONNECT* (Seção 3.2) e *PUBLISH* (Seção 3.2) são tratados, que é o escopo deste trabalho.

3.2 PACOTE TIPO *CONNECT*

Este deve ser o primeiro pacote à ser enviado por um cliente a um servidor. O seu cabeçalho variável é composto por 10 *bytes*.

3.2.1 Cabeçalho variável do pacote *CONNECT*

Tabela 7: Formato do cabeçalho variável do pacote tipo *CONNECT*

	Descrição	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Nome do protocolo									
byte 1	Tamanho MSB (0)	0	0	0	0	0	0	0	0
byte 2	Tamanho LSB (4)	0	0	0	0	0	1	0	0
byte 3	‘M’	0	1	0	0	1	1	0	1
byte 4	‘Q’	0	1	0	1	0	0	0	1
byte 5	‘T’	0	1	0	1	0	1	0	0
byte 6	‘T’	0	1	0	1	0	1	0	0
Camada do protocolo									
byte 7	Camada (4) TCP/IP	0	0	0	0	0	1	0	0
Sinalizadores de conexão (flags)									
byte 8	User Name (1) Password (1) Will Retain (0) Will QoS (01) Will Flag(1) Clean Session (1) Reserved (0)	1	1	0	0	1	1	1	0
Keep Alive									
byte 9	Keep Alive MSB(0)	0	0	0	0	0	0	0	0
byte 10	Keep Alive LSB(10)	0	0	0	0	1	0	1	0

Fonte: *MQTT Version 3.1.1* (BANKS; GUPTA. 2014. p. 23)

Pode-se observar na Tabela 7 que os *bytes* de 1 a 6 descrevem o nome do protocolo, é utilizado por programas de verificação de pacotes de rede para identificar o nome do protocolo que está trafegando. O servidor pode recusar a conexão caso o conteúdo deste campo seja diferente do padrão; O *byte* 7 indica que o protocolo é de aplicação; O *byte* 8 em seus *bits* 6 e 7 fornecem os parâmetros básicos para autenticação de usuário; E os *bytes* 9 e 10 representam o tempo máximo de vida de um pacote.

3.2.2 Carga do pacote *CONNECT*

Contém um ou mais campos com tamanhos pré-fixados, a existência de um campo está condicionada aos sinalizadores (*flags*) do cabeçalho variável. Os campos necessariamente devem aparecer na ordem, Identificador do cliente, *Will Topic, Will Message, User Name* e *Password*.

• Identificador do cliente – Cada cliente deve ter um único identificador, denominado de ClientId, este campo é utilizado para controlar e manter a sessão entre o cliente e o servidor.

• Will Topic – Neste campo deve-se indicar o tópico no qual a mensagem será publicada.

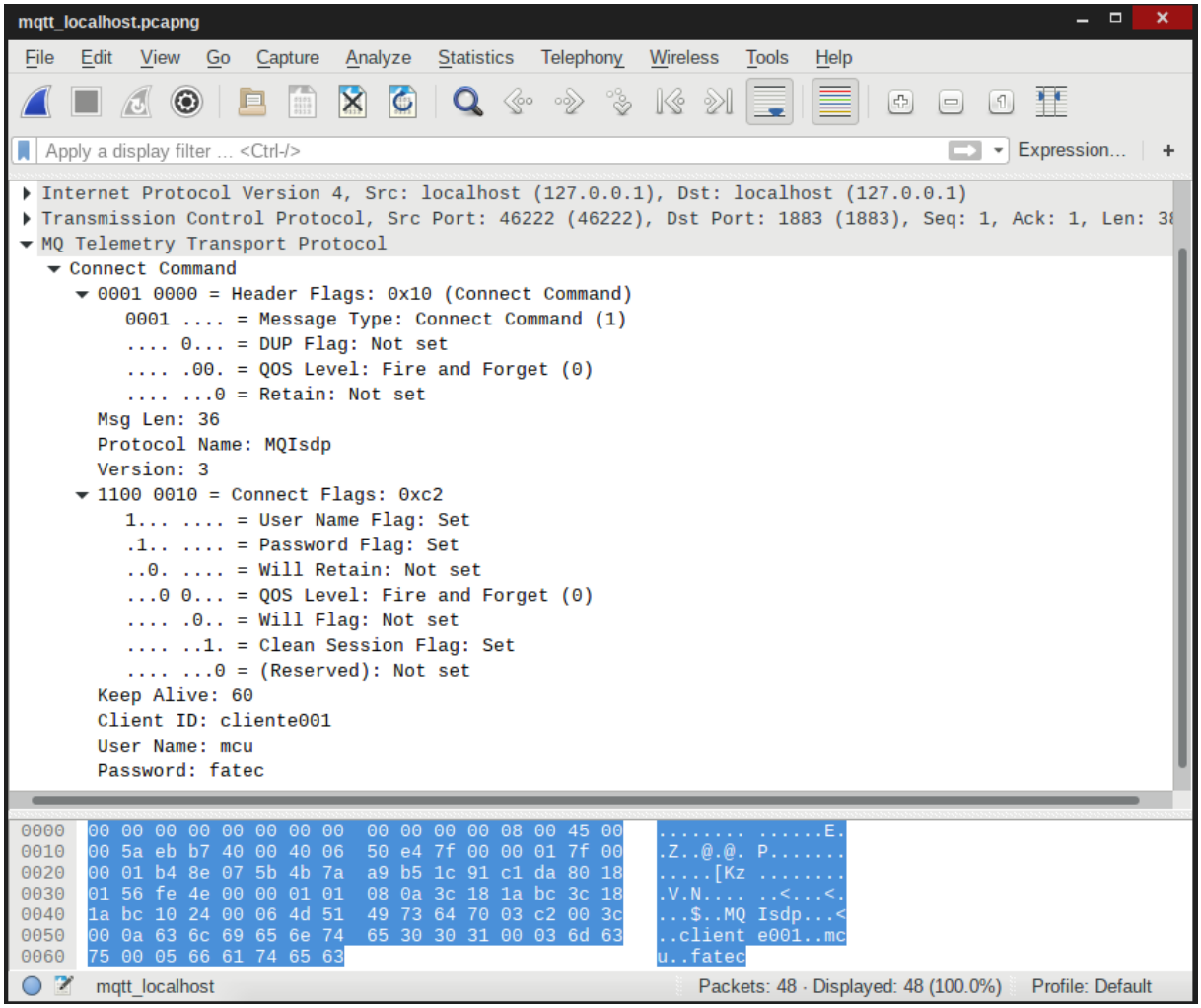
• Will Message – Mensagem a ser publicada.

• User Name – Nome do usuário, o servidor pode utilizar este nome para realizar a autenticação e autorização do usuário.

• Password – Define a senha do usuário. A senha é transmitida em texto claro, não existe nenhum mecanismo de proteção oferecido pelo protocolo com a finalidade de proteção da senha.

3.2.3 Captura e identificação de um pacote do tipo *CONNECT*

Figura 11 - Pacote do tipo *CONNECT*



Fonte: *Printscreen* da tela do *software Wireshark*

Pacote capturado no momento de solicitação de conexão com o *Broker.* O pacote do tipo *CONNECT* (Seção 3.2) é a primeira mensagem a ser enviada pelo cliente ao servidor. Na Figura 11 é possível identificar os cabeçalhos e os campos do pacote *MQTT* conforme as Tabelas 3, 4, 5, 6 e 7, o *software Wireshark* já separa os *bytes* conforme a especificação do protocolo e os traduz de forma a facilitar a leitura.

3.3 PACOTE TIPO *PUBLISH*

O pacote do tipo *PUBLISH* é responsável por transportar uma mensagem de aplicação.

O cabeçalho fixo deste pacote contém os *bits* 00110000 identificando o seu tipo.

3.3.1 Cabeçalho variável do pacote *PUBLISH*

É de responsabilidade do cabeçalho variável indicar o tópico de publicação. A Tabela 8 exemplifica um tópico denominado de a/b (*bytes* 3, 4 e 5 da Tabela 8).

Os *bytes* 1 e 2 (Tabela 8) contém o tamanho, em caracteres, do tópico de publicação. Cada caractere é representado por um *byte* na codificação *UTF-8*. Logo após o tópico de publicação estão 2 *bytes* (*bytes* 6 e 7 Tabela 8*)* que contém o valor numérico do identificador do pacote. Este identificador é utilizado pelo *broker* para gerenciamento de seção de comunicação.

Tabela 8: Formato do cabeçalho variável do pacote tipo *PUBLISH*

BYTES	**DESCRIÇÃO**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Nome do tópico									
byte 1	Tamanho MSB (0)	0	0	0	0	0	0	0	0
byte 2	Tamanho LSB (3)	0	0	0	0	0	0	1	1
byte 3	‘a’ (0x61)	0	1	1	0	0	0	0	1
byte 4	‘/’ (0x2F)	0	0	1	0	1	1	1	1
byte 5	‘b’ (0x62)	0	1	1	0	0	0	1	0
Indentificador do pacote									
byte 6	MSB(0)	0	0	0	0	0	0	0	0
byte 7	LSB(10)	0	0	0	0	1	0	1	0

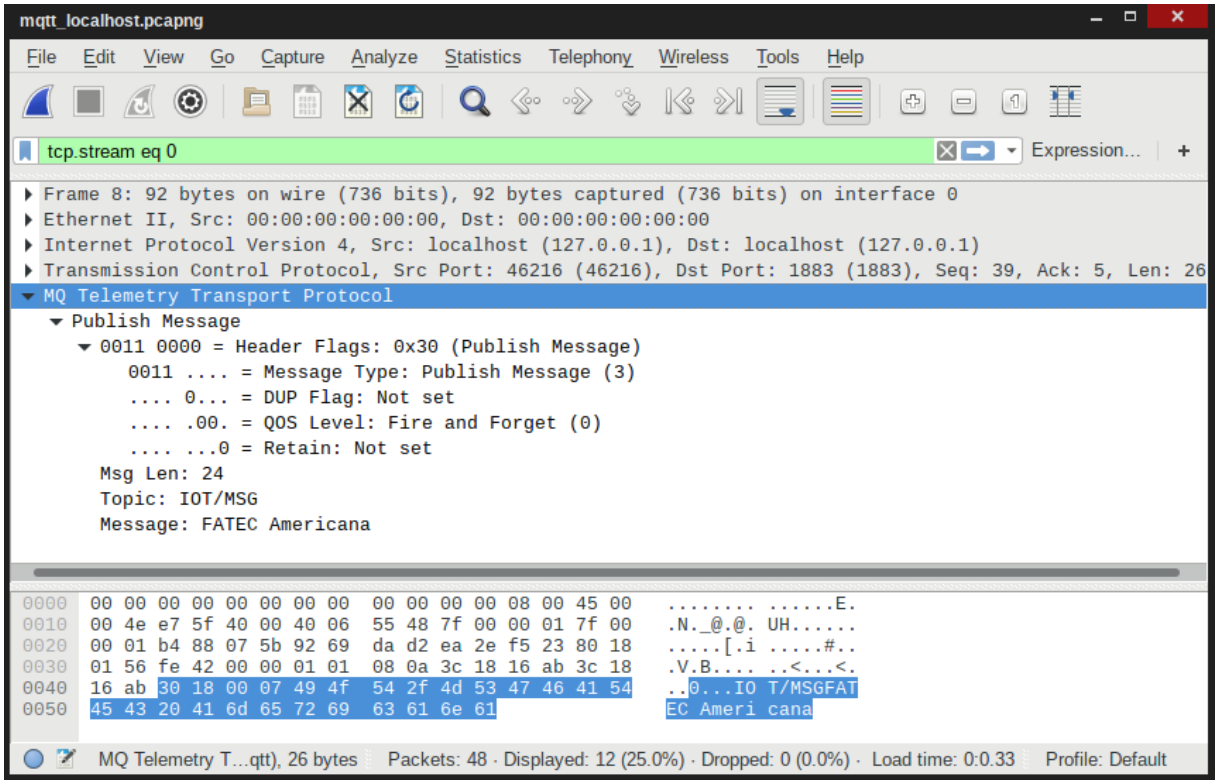
Fonte: *MQTT Version 3.1.1* (BANKS; GUPTA. 2014. p. 29)

3.3.2 Carga (*Payload*) do pacote *PUBLISH*

Contém a mensagem que deve ser publicada. O tamanho do campo pode ser calculado subtraindo o tamanho do cabeçalho variável do tamanho do campo comprimento restante *(Remaining Length)* do cabeçalho fixo.

3.3.3 Captura e identificação de um pacote do tipo *PUBLISH*

Figura 12 - Pacote do tipo *PUBLISH*



Fonte: *Printscreen* da tela do *software Wireshark*

Na Figura 12 foi selecionado e destacado na cor azul o pacote *MQTT. O software* exibe no quadro inferior os *bytes* capturados de forma bruta. A seleção também se expande para esta caixa, o tamanho total deste pacote *MQTT* é de 26 *bytes,* valor indicado na barra inferior de *status.* Percebe-se que o valor do campo tamanho restante (*Remaining Length)* aqui denominado de *Msg Len* é de 24 *bytes*, ou seja, é o valor total de 26 *bytes* subtraído o valor de 2 *bytes* referente ao tamanho do cabeçalho fixo. Também é possível ler em texto claro a mensagem a ser publicada e o tópico para publicação.

A identificação dos elementos estruturais do pacote *MQTT* e a leitura de seu conteúdo só é possível devido ao não uso do protocolo criptográfico *TLS*. O Capítulo 4 exemplifica um pacote sem uso de TLS (Figura 15) e um pacote com uso de TLS (Figura 23).

4 PROTÓTIPO

O ambiente de teste e análise de dados utiliza equipamentos reais, ou seja, não foi aplicada a técnica de simulação ou virtualização de *hardware*. É composto basicamente por um dispositivo de recursos limitados, denominado de sensor, um computador pessoal e um equipamento de interconexão de rede sem fio.

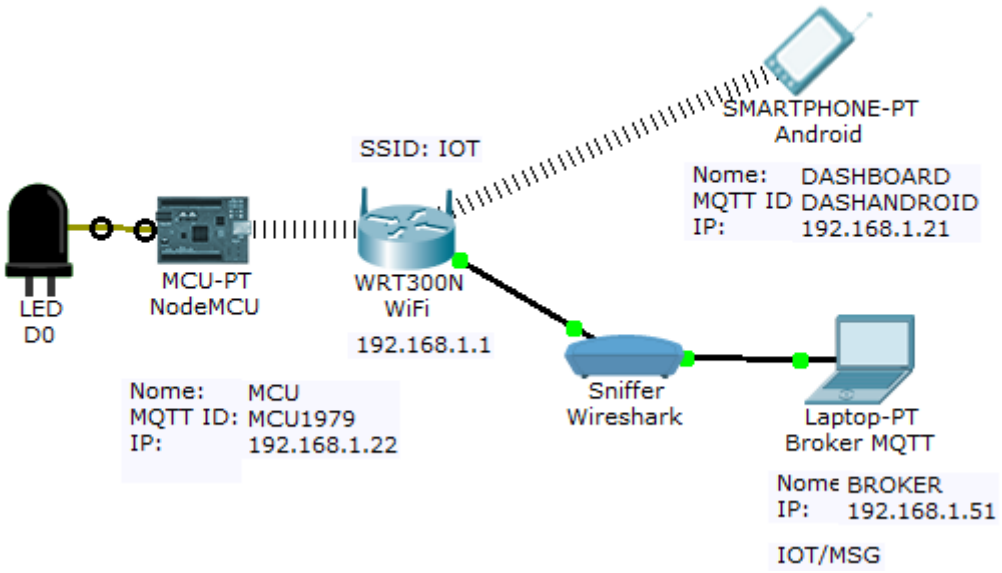
O objetivo do protótipo é proporcionar uma rede *IoT* para observar o comportamento e o tempo gasto pelo sensor durante a tarefa de codificação e publicação de mensagens.

Foi desenvolvido um *software* utilizando a ferramenta oficial da plataforma do *Arduino* em conjunto com as bibliotecas disponíveis e compatíveis com o hardware do sensor. A unidade de medida convencionada para a geração e análise dos dados em relação ao tempo é microssegundos (us).

São apresentados dois cenários para a coleta de dados. No primeiro as mensagens são publicadas em texto claro, sendo assim vulnerável a alguns tipos de ataques. No segundo cenário utiliza o protocolo *TLS* que insere uma camada de criptografia para proteger a comunicação. A técnica de criptografia aplicada ao protocolo *TLS* requer recursos computacionais adequados, o que representa um desafio para a segurança da informação no ambiente *IoT* que é composto por vários dispositivos de recursos limitados (No ambiente de teste o sensor é o objeto de análise).

4.1 DIAGRAMA DA REDE *IOT*

Figura 13 - Rede *IoT*



Fonte: próprio autor

A Figura 13 mostra o modo como foram interligados os equipamentos utilizados no protótipo. O padrão de comunicação se assemelha ao modelo *Device-to-Cloud* (Figura 2) proposto pela *RFC 7452.* O comportamento do servidor (*broker*) independe do fato de estar na “nuvem” ou presente na rede local.

4.2 EQUIPAMENTOS E SOFTWARES UTILIZADOS

A Tabela 9 lista o *Hardware* utilizado e suas características básicas, para cada *Hardware* foram listados os *softwares* utilizados em cada. A diferença entre as capacidades e desempenho ficam evidentes. O *hardware* do Sensor é o mais limitado, composto por um processador de 80MHz e 4MB de memória.;

Tabela 9: *Hardware* e *Software* utilizados

TIPO	**FABRICANTE**	**MODELO**	**PROCESSADOR**	**MEMÓRIA**
Notebook	HP	G42	AMD Phenom™ II. Triple-Core Mobile N850. 2,20 GHz	8GB
	Softwares	- Sistema Operacional Linux Slackware versão 14.2 64bits[3]		

	utilizados:	- *Eclipse Mosquitto,* instalado o pacote completo: *broker*, *publisher* e *subscriber (Protocolo MQTT)*[4] - Arduino IDE[5] - Wireshark[6]		
Sensor	WEMOS Eletronics	Mini D1	ESP-8266EX 80MHz	4MB
	Softwares	- Foram desenvolvidos dois softwares, um para cada cenário proposto: TCC_NodeMCU_C01.ino e TCC_NodeMCU_C02.ino.		
WiFi	TP-Link	WDR-4300	Atheros AR9344 560MHz	128MB

Fonte: próprio autor.

Todos os softwares utilizados estão disponíveis para download de forma gratuita.

4.3 SOFTWARE DESENVOLVIDO

Dois softwares são desenvolvidos, um para cada cenário proposto. O objetivo é gerar amostras de dados em formato de fácil interpretação. Codificado em linguagem estilo C, padrão da *IDE* do *Arduino.* Faz-se necessário adicionar bibliotecas específicas: *PubSubClient*[7] e *ESP8266FS*[8]*,* também são utilizadas outras bibliotecas para incorporar funções de acesso remoto e monitoramento do software em tempo de execução, estas bibliotecas não estão diretamente relacionadas ao resultado final do *software e* não são abordadas neste trabalho.

Os nomes dados aos códigos fontes são: *TCC_NodeMCU_C01.ino (Anexo C)* para o cenário 1 e *TCC_NodeMCU_C02.ino* (Anexo D) para o cenário 2, eles se diferem somente pelo acréscimo de criptografia ao código do cenário 2. As porções de interesse do software são abordadas em conjunto com a descrição e análise dos cenários 1 e 2 (seções 4.5.1.1 e 4.5.2.3 respectivamente).

---

3 *Slackware Linux*. Disponível em <http://www.slackware.com/>. Acesso em 01 de jun. 2018.

4 *Eclipse Mosquitto*. Disponível em <https://mosquitto.org/>. Acesso em 02 de jun. 2018.

5 *Arduino IDE.* Disponível em <https://www.arduino.cc/>. Acesso em 02 de jun. 2018.

6 *Wireshark*. Disponível em <https://www.wireshark.org/>. Acesso em 02 de jun. 2018.

7 *Arduino Library List. PubSubClient*. Disponível em <https://www.arduinolibraries.info/libraries/pub-sub-client>. Acesso em 2 mai. 2018.

8 ESP8266 Arduino Core. ESP8266FS. Disponível em <https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.1.3/ESP8266FS-0.1.3.zip>. Acesso em 2 mai. 2018.

De forma simplificada pode-se resumir o algoritmo de geração da amostra, da seguinte maneira:

tempo = *timestamp*
publica a mensagem MQTT
tempo = *timestasmp* – tempo
publica a mensagem com o valor do tempo
repete os passos anteriores 1000 vezes

Onde o *timestamp* representa o tempo em microssegundos a partir do momento em que o sensor foi energizado.

No caso do cenário 2 as publicações são criptografadas. Depara-se com o problema de sincronização do relógio. É necessário sincronizar os relógios para o funcionamento do modelo de verificação de certificados e criptografia atrelados ao protocolo *TLS*.

4.4 METODOLOGIA DE COLETA DE DADOS

Os dados são gerados pelo *software* desenvolvido especialmente para esta função (seção 4.3), este software publica mensagens contabilizando uma amostra de 1000 pacotes enviados de forma sequencial. Para cada envio é registrado o tempo em microssegundos do empacotamento e publicação do mesmo sendo o número sequencial do pacote e o tempo registrado publicados em uma mensagem subsequente. Estes valores são separados pelo caractere “;” facilitando a exportação dos dados para uma planilha de cálculo e confecção do gráfico.

Em paralelo à geração das amostras, todo o tráfego da rede é registrado de forma bruta para a criação de estatísticas, permitindo comparar o desempenho do ambiente de rede antes e depois do uso de criptografia.

O número sequencial do pacote publicado é utilizado apenas para efeito de organização dos dados.

A amostra padronizada possui a seguinte característica:

41;1138

O primeiro valor é o número sequencial e o segundo valor é o tempo em microssegundos. Os Anexos A e B contêm uma amostra dos dados coletados.

4.5 CENÁRIOS PROPOSTOS

Dois cenários foram elaborados, onde cada um gerou uma amostra de dados, as amostras foram analisadas e os resultados expostos individualmente em gráficos de dispersão. Também foi confeccionado um gráfico contendo as duas amostras possibilitando uma análise comparativa.

4.5.1 Cenário 1 – Publicação de pacotes em texto claro

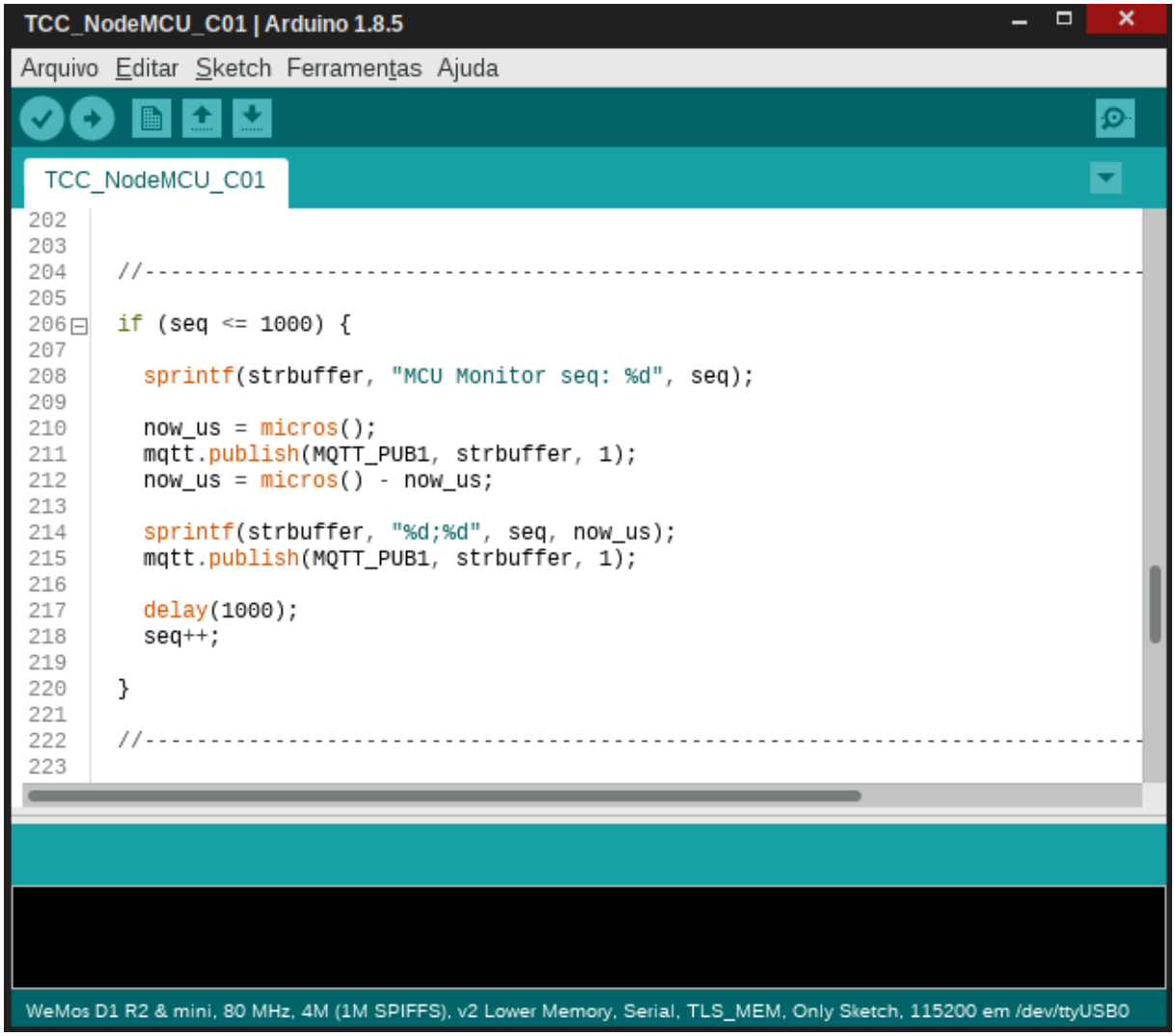
O objetivo do Cenário 1 é a publicação de mensagens de texto claro pelo sensor com destino ao *Broker* utilizando o protocolo *MQTT*.

Criação e registro da amostra

Executado o software elaborado TCC_NodeMCU_C01.ino[9] no dispositivo *nodeMCU*, o sensor, as publicações são geradas conforme o trecho de código exibido na Figura 14.

9 TCC_NodeMCU_C01.ino, Disponível na íntegra no Anexo C deste trabalho.

Figura 14 - Cenário 1 – Trecho do Código - Publicação de mensagens em texto puro



Fonte: *Printscreen* da *IDE* do *Arduino*, código fonte TCC_NodeMCU_C01

Para este primeiro cenário não foi necessário realizar ajustes de configuração no *broker mosquitto* e o executamos em sua configuração padrão. Os pacotes de rede foram aceitos pelo *broker* e transmitidos em texto puro.

Observem que sempre são publicadas duas mensagens, o tempo de empacotamento e envio da primeira mensagem é registrado, calculado e o resultado é enviado na segunda mensagem que contém o número sequencial, formando o par de dados (número sequencial; tempo). É respeitado um tempo de intervalo de um segundo para o registro de envio da próxima sequência. Este par de dados é base para a formação da amostra analisada. O registro da amostra foi obtido por um cliente *MQTT* inscrito na lista *IOT/MSG* que ao receber uma publicação desvia a saída em tela para um arquivo. O *script linux* coleta_cenario01.sh utilizado para esta

tarefa está contido no Anexo E.

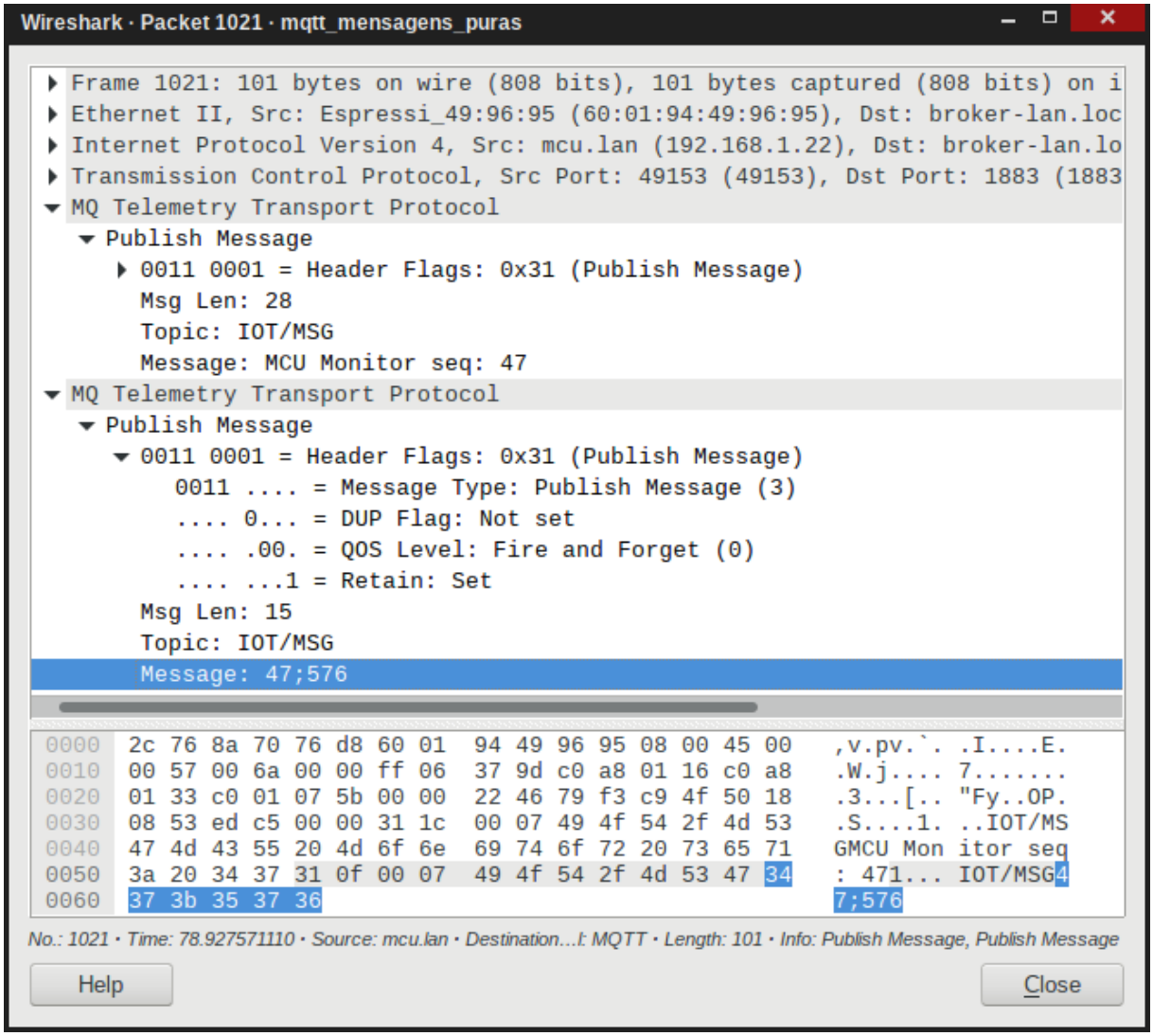
O resultado é um arquivo contendo a amostra de 1000 mensagens (um trecho desta amostra pode ser observada no Anexo A). Os pares de valores separados por ponto e vírgula foram exportados e analisados em uma planilha de cálculos.

Captura dos pacotes transmitidos pela rede

Paralelamente ao processo de criação da amostra, foi utilizado o *software Wireshark* para registrar de forma bruta todos os pacotes trafegados pela rede. Este registro permite observar o conteúdo de cada pacote e confirmar que os mesmos, para o cenário 1, foram transmitidos pela rede em forma de texto puro, ou seja, sem criptografia.

O foco da análise é apenas a estrutura do protocolo de aplicação *MQTT*.

Figura 15 - Cenário 1 – *Wireshark* - Análise bruta de um pacote



Fonte: *Printscreen* do *software Wireshark* no *Linux*

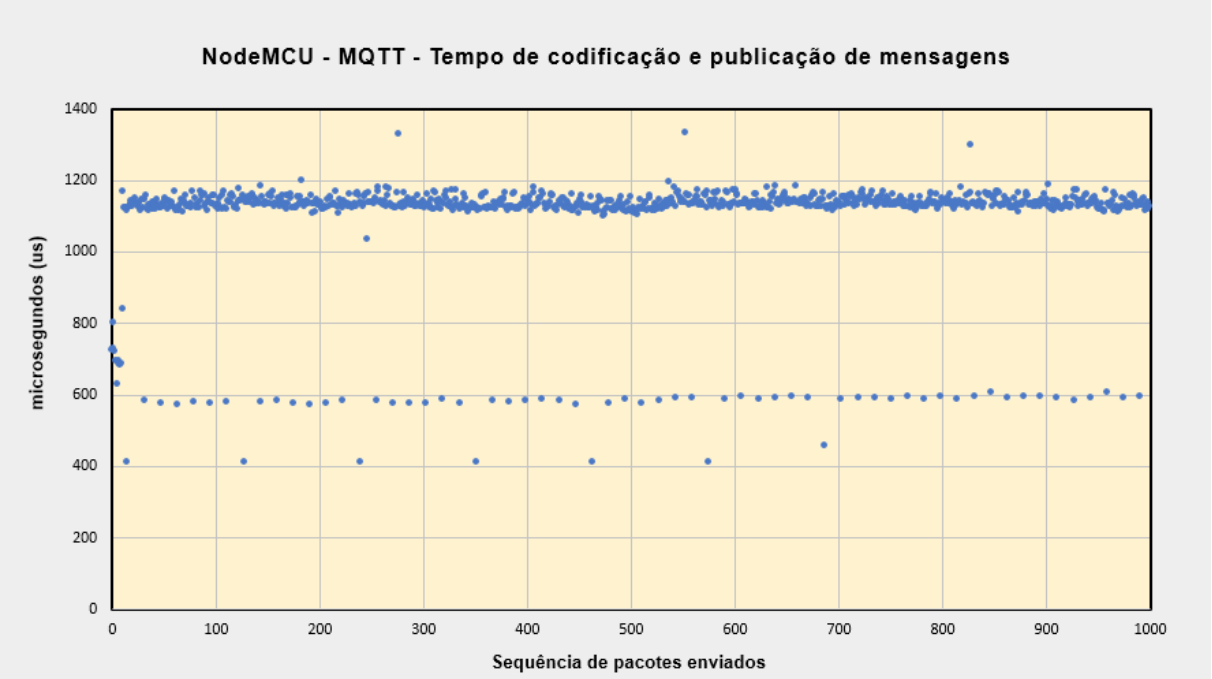
Pode-se observar na Figura 15 de forma bem clara o conteúdo da mensagem a ser publicada, o tópico para publicação, o tipo do pacote *MQTT* e todos os *flags* associados.

Neste registro do pacote em específico, ocorreu o fato de que em um único pacote *TCP* fossem transportados dois pacotes *MQTT.* Foi notado que este detalhe ocorre sempre quando o tempo de processamento dos pacotes é pequeno, menor que a metade da média registrada, assim, explica-se pela própria característica das camadas do protocolo do *TCP/IP* que durante o empacotamento da camada 4 (camada de aplicação) houve espaço no *buffer* e tempo para que o protocolo *MQTT* unisse os dois pacotes *MQTT* e o enviesse para a camada 3, no caso ao protocolo *TCP*, de maneira transparente, ou seja, a camada *TCP* não sabe se está

transportando uma ou duas mensagens *MQTT*.

Análise da amostra

Figura 16 - Cenário 1 - Representação visual da amostra gerada



Fonte: próprio autor

O gráfico de dispersão reúne todas as publicações de forma sequencial em seu eixo horizontal e em seu eixo vertical é exibido o tempo de processamento e envio de cada publicação pelo sensor.

Três informações importantes foram observadas:

Tempo médio: 1098µs

Tempo máximo: 1332µs

Tempo mínimo: 409µs

Para o sensor que é um dispositivo de recursos limitados, com uma velocidade de processamento de 80MHz e conectado a uma rede *WiFi WPA2*, o tempo médio calculado foi de 1098µs o que é considerado satisfatório ao objetivo proposto. Não houve perda de nenhuma mensagem transmitida.

Problemas identificados envolvendo segurança da informação

As informações foram capturadas e interpretadas da forma mais simples possível, neste aspecto abre margem para um ataque passivo onde somente as informações são interessantes ao atacante ou um ataque de interceptação e injeção de dados. Também não existem controles por parte do *broker MQTT* de autenticação de clientes e nem de restrições de publicações, mas isso só ocorreu pelo fato de utilizarmos a configuração básica que por padrão não vem com os itens de segurança habilitados.

Desta forma não é possível se obter a confidencialidade, integridade e disponibilidade da informação e nem como garantir o não repúdio.

Os únicos motivos de se utilizar o protocolo *MQTT* sem aplicar as regras de segurança é para a realização de testes em ambientes controlados, por compatibilidade com dispositivos que não apresentam tais possibilidades, ou de forma consciente em um ambiente no qual toda a responsabilidade de segurança está atrelada a outros equipamentos, como um *Firewall* por exemplo.

4.5.2 Cenário 2 – Publicando mensagens criptografadas

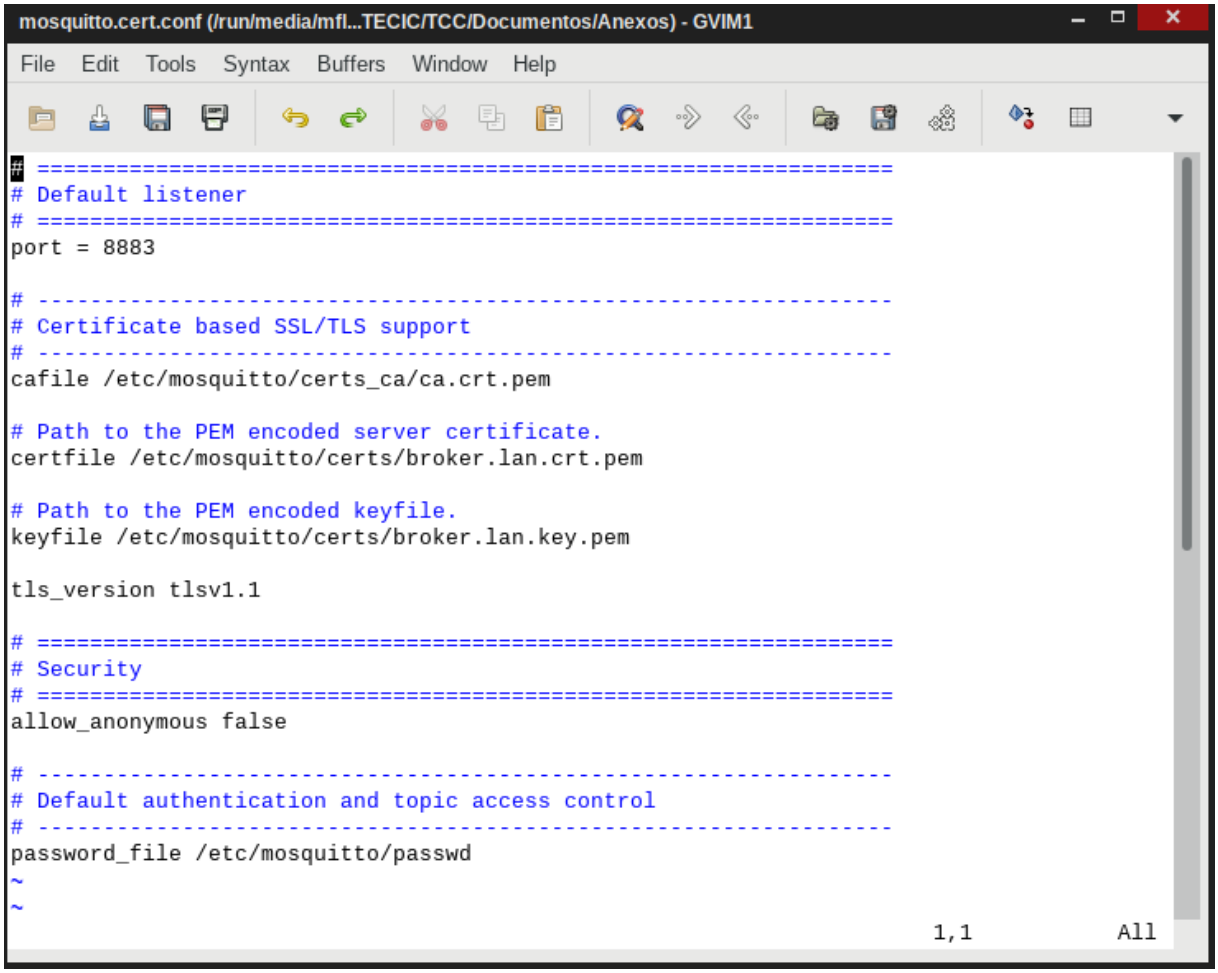
Neste cenário o sensor realizou a publicação de mensagens de forma criptografada utilizando o esquema de certificação digital e autenticação de usuário.

Para utilizar o uso de criptografia é necessário realizar algumas alterações de configuração no servidor *broker mosquitto*, além disso, foram criados certificados digitais. Todos estes processos são descritos a seguir.

Ajustes de configuração do broker

Foram realizadas alterações no arquivo de configuração do *broker mosquitto*. Cada ação executada aumentou o nível de segurança em relação aos princípios da segurança da informação. O nome do arquivo de configuração modificado é mosquitto.cert.conf e está localizado na pasta /etc/mosquitto no sistema operacional Linux. Os parâmetros alterados estão expostos na Figura 17 e são as alterações mínimas para possibilitar uma abordagem mais segura do protocolo.

Figura 17 - Cenário 2 – Configurações do *broker mosquitto*



Fonte: Arquivo de configuração do *software mosquitto (mosquitto.cert.conf)*

Cada parâmetro alterado está descrito e relacionado aos princípios de *SI* na Tabela 10.

Tabela 10: Relação parâmetro de configuração do *mosquitto* e princípio de *SI* envolvido.

PARÂMETRO	***DESCRIÇÃO***	***PRINCÍPIO DE SI***
port	A porta 8883 é a porta recomendada para o uso do protocolo com certificados e criptografia, difere da porta convencional 1883.	Não há ganho neste aspecto. Embora permitir o uso de uma qualquer porta em associação a um protocolo encapsulado por criptografia pode dificultar a identificação do serviço provido, em relação a uma porta padrão.
cafile	Define o caminho para o arquivo contendo o certificado CA, este é o certificado de	AUTENTICIDADE

	confiança que deve assinar os outros certificados: do broker e dos seus clientes.	
certfile	Indica o caminho para o certificado do broker no qual sua veracidade foi atestada pelo certificado CA indicado no parâmetro cafile.	AUTENTICIDADE INTEGRIDADE
keyfile	Arquivo contendo a chave privada para o broker. Esta chave foi utilizada para gerar a requisição de assinatura do certifico.	AUTENTICIDADE
tls_version	Indica a versão do protocolo TLS a ser utilizado. O *listener* (estado do servidor de aguardar uma conexão à uma porta TCP) não aceitará conexões não criptografadas.	CONFIDENCIALIDADE INTEGRIDADE

Fonte: próprio autor

Certificados digitais

O uso de certificado digital exige que cada equipamento ou dispositivo obtenha o seu próprio certificado assinado por uma entidade na qual pode-se confiar. Esta entidade pode ser uma entidade certificadora que faz parte de uma infra-estrutura de chaves públicas (*PKI*) ou o uso de um certificado autoassinado que, para fins deste protótipo, funciona de maneira idêntica. A segunda opção foi a utilizada e todos os certificados foram criados localmente utilizando o *software OpenSSL* no ambiente *Linux.*

A primeira etapa é criar um certificado do tipo *CA*, este certificado fica em posse de uma entidade confiável e é ele quem assina a veracidade dos outros certificados bem como realiza o processo de verificação da autenticidade de sua assinatura.

O servidor *broker* utiliza este certificado para validar os certificados dos seus clientes. Ele próprio tem que ter um certificado assinado para proceder com o processo de troca de chaves com os seus clientes.

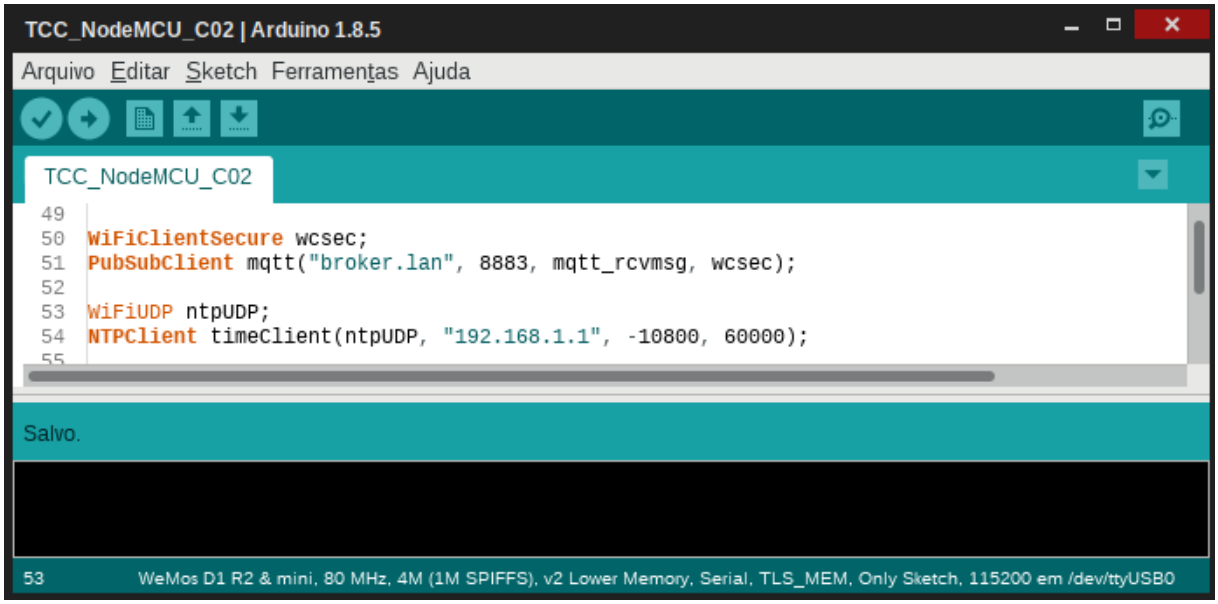
Para o sensor foi criado um terceiro certificado assinado.

O processo de criação dos certificados e os comandos utilizados estão listados no Anexo F.

Criação e registro da amostra

O *software* TCC_NodeMCU_C01.ino foi alterado de maneira a incluir o uso de criptografia dando origem a versão TCC_NodeMCU_C02.ino (listagem integral do código fonte disponível no Anexo D). As principais modificações são exibidas na Figura 18 e discutidas a seguir.

Figura 18 - Cenário 2 – Objetos seguros



Fonte: *Printscreen* da *IDE* do *Arduino*, código fonte TCC_NodeMCU_C02

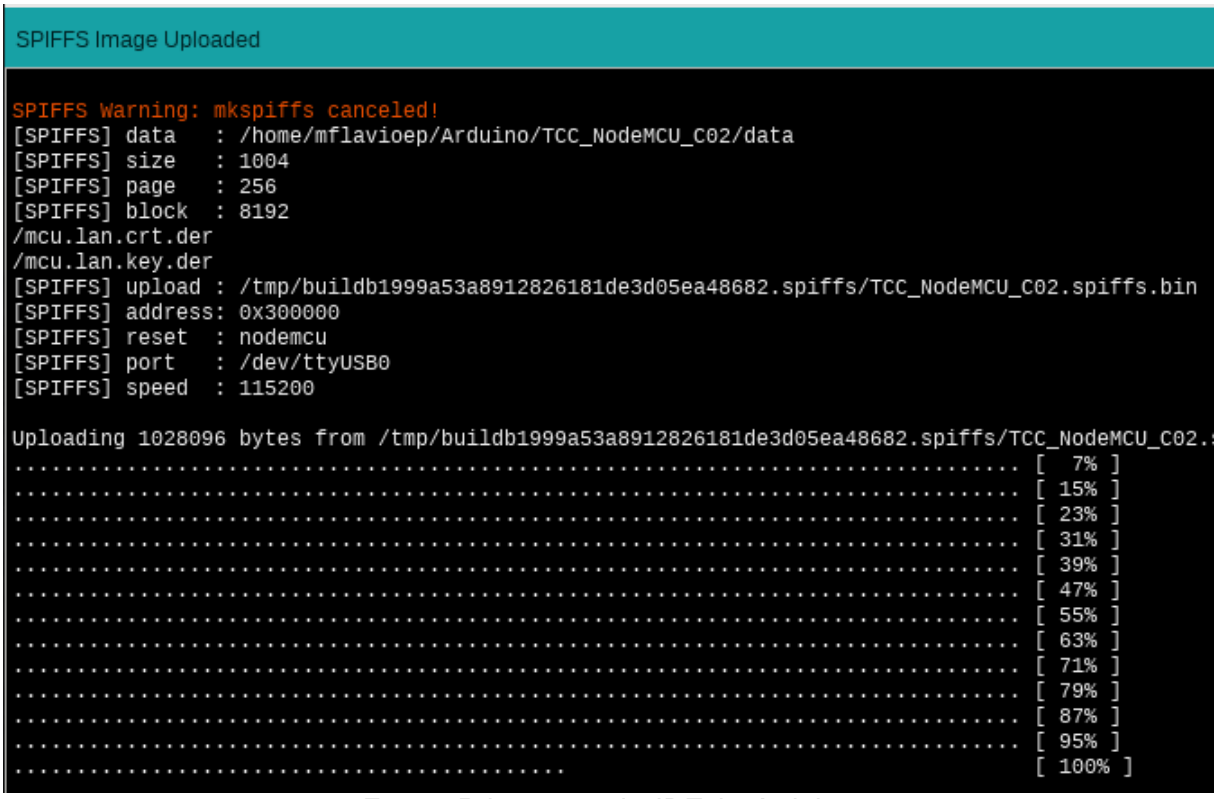
A classe que implementa o protocolo *MQTT* para o *nodeMCU* é a *PubSubClient* (linha 51). O quarto parâmetro de instanciação é o objeto que implementa a conexão com a rede sem fio. No cenário 1 foi utilizado o objeto *WiFiClient* e para este cenário, ela foi substituída pelo objeto da classe *WiFiClientSecure* (linha 50). A porta de conexão com o *broker* também foi alterada, de 1883 para 8883 (esta é a porta recomendada para a publicação de mensagens criptografadas).

Outro detalhe importante é que para possibilitar a verificação da validade dos certificados e o processo de criptografia, faz-se necessário sincronizar as datas e relógios dos equipamentos. Para isto foi utilizado um servidor *NTP* provido nativamente pelo dispositivo *WiFi* (Roteador *TP-Link*) (este dispositivo também provê o servidor *DHCP)*.

O sensor não apresenta um sistema de arquivos de forma nativa. Para

copiar os arquivos do certificado (certificado e chave privada) é necessário instalar a biblioteca ESP8622FS, esta reserva um espaço em memória (1MB) no dispositivo para que seja tratado como sistema de arquivos. Os certificados devem estar em sua forma binária, conhecido por certificados DER. A transferência é realizada pela *IDE* do *Arduino,* conforme visto na Figura 19.

Figura 19 - *Cenário 2* - *Upload* do certificado para o sensor



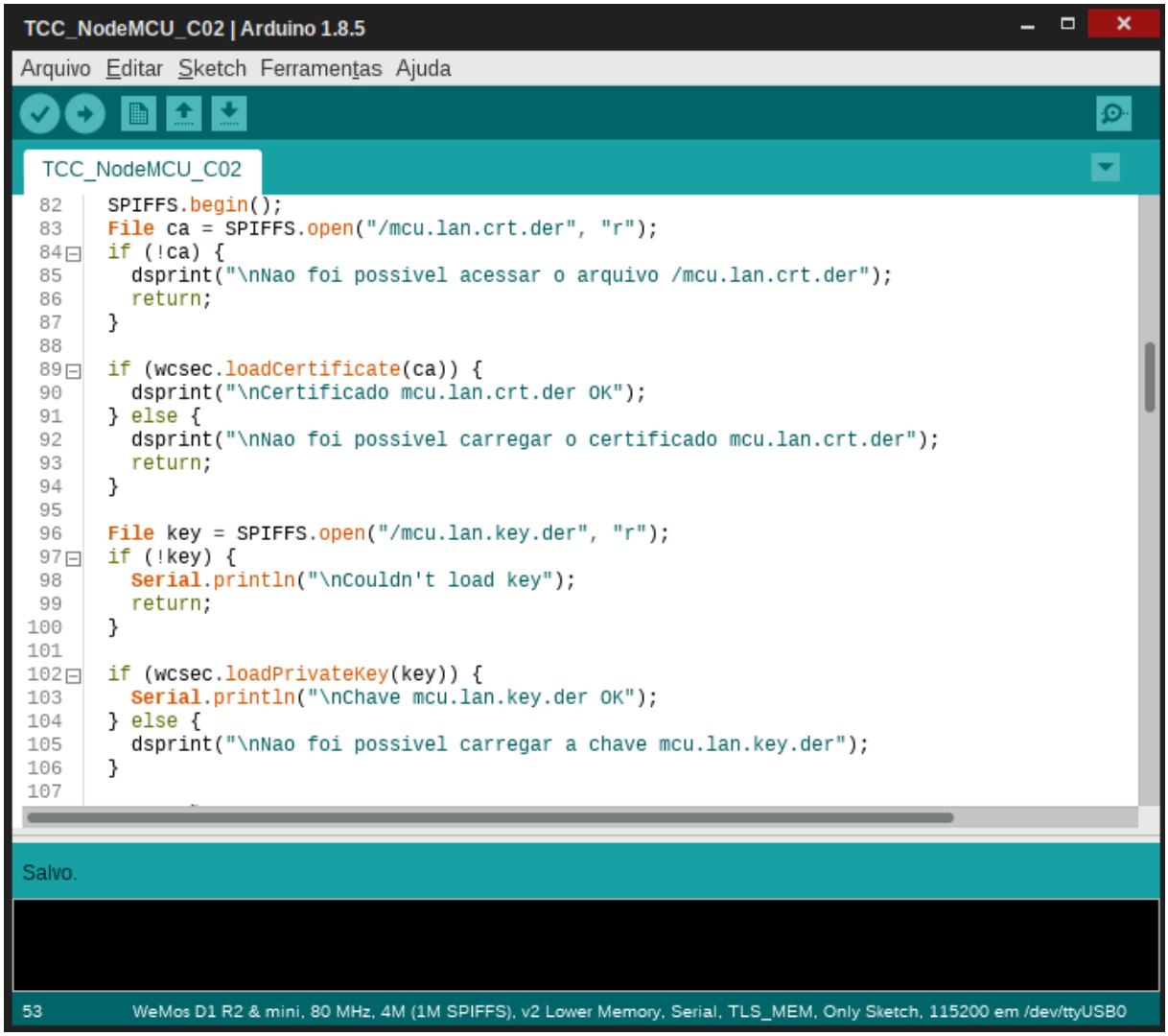
Fonte: *Printscreen* da *IDE* do *Arduino*

Na Figura 19, pode-se observar que todos os arquivos contidos na pasta local do projeto do *software* são copiados e agrupados em um único arquivo binário temporário. Logo após, o arquivo binário é copiado para uma porção de memória do sensor que será utilizada pelo sistema de arquivos.

Agora é possível abrir os arquivos em tempo de execução do *software*.

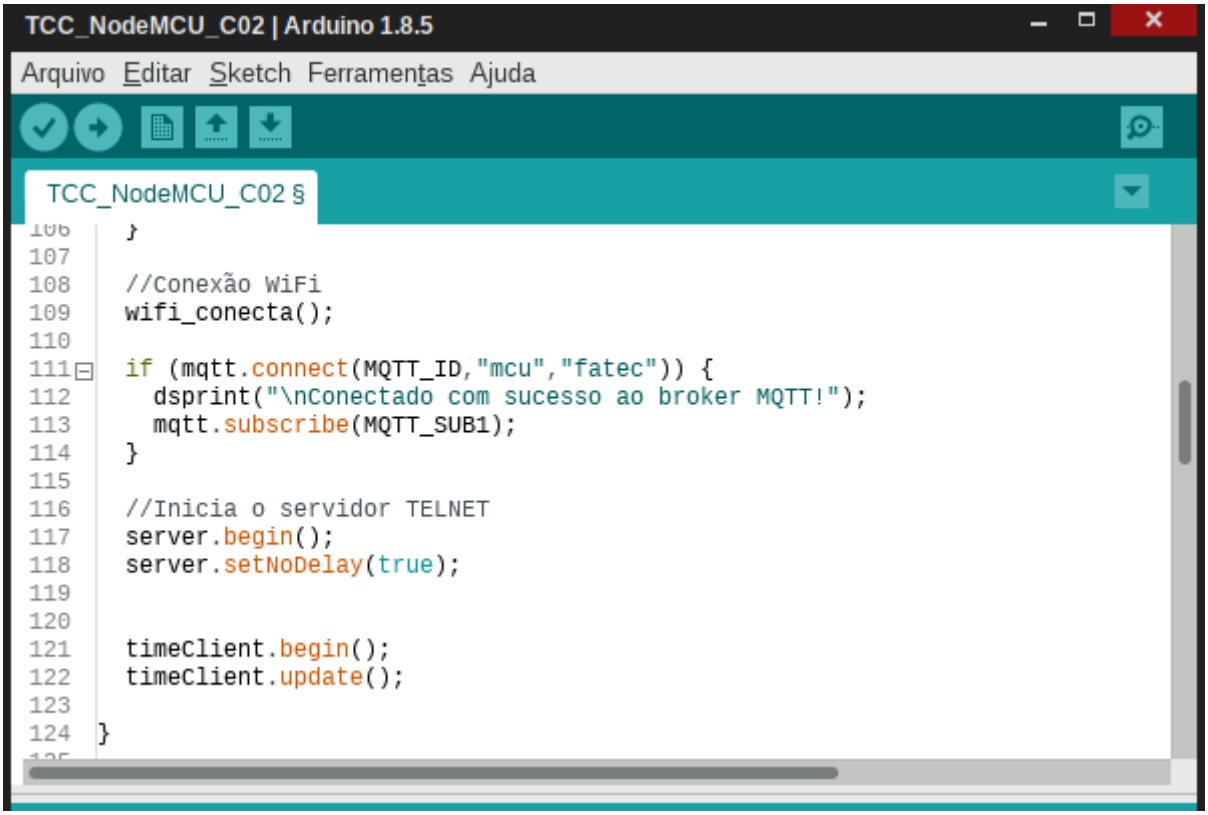
O trecho de código da Figura 20 demonstra o modo de abertura dos arquivos pela classe *SPIFSS* (linhas 82,83 e 96), estes arquivos são utilizados pela classe *WifiClientSecure* para carregar o certificado e a chave privada (linhas 89 e 102) que são os requisitos necessários para habilitar o uso do *TLS*.

Figura 20 - Cenário 2 – Abertura e Carregamento do certificado



Fonte: *Printscreen* da *IDE* do *Arduino*

Figura 21 - Cenário 2 – Conexão *MQTT* com usuário e senha



Fonte: *Printscreen* da *IDE* do *Arduino*

A linha 111 do trecho de código exibido na Figura 21, comando *connect* é o local onde especifica-se o usuário e senha a serem utilizados para a validação do usuário (usuário “mcu” com a senha “fatec”).

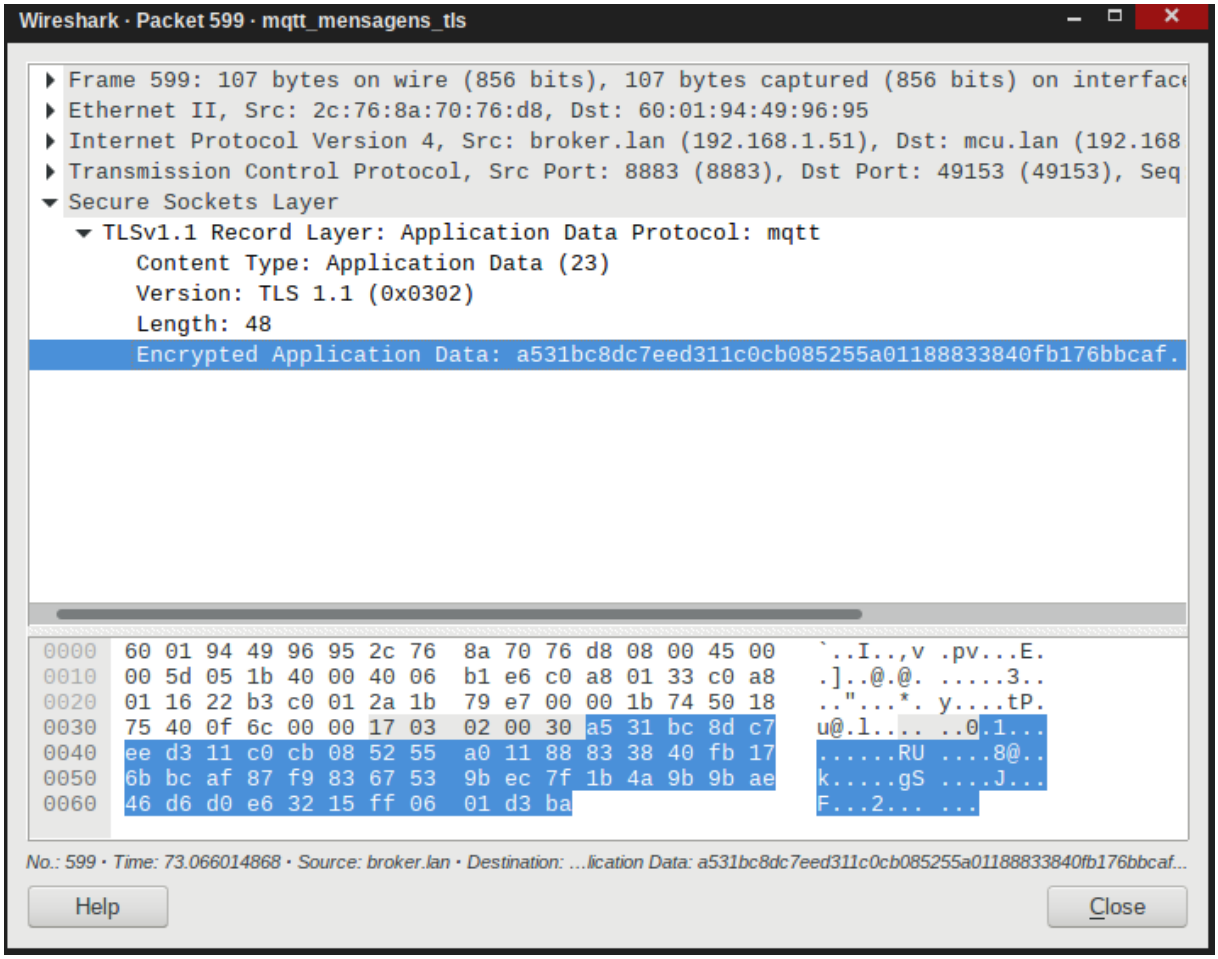
Figura 22 - Cenário 2 - Publicação de mensagens em texto cifrado

Fonte: *Printscreen* da *IDE* do *Arduino*

O trecho do código da publicação das mensagens não difere do cenário 1. Toda a configuração foi realizada nos trechos anteriores. A sequência de conexão e publicação da mensagem, quando analisada de forma bruta (pacotes trafegados pela rede), evidencia o processo de criptografia.

Captura dos pacotes transmitidos pela rede

Figura 23 - Cenário 2 – *Wireshark* - Análise bruta de um pacote



Fonte: *Printscreen* da tela do *software Wireshark*

Nota-se na Figura 23 que ao publicar uma mensagem no cenário 2, o protocolo *MQTT* foi encapsulado pelo protocolo *TLS*, ambos os protocolos são da camada de aplicação *TCP/IP,* conforme explicitado na seção 2.4.3.1.

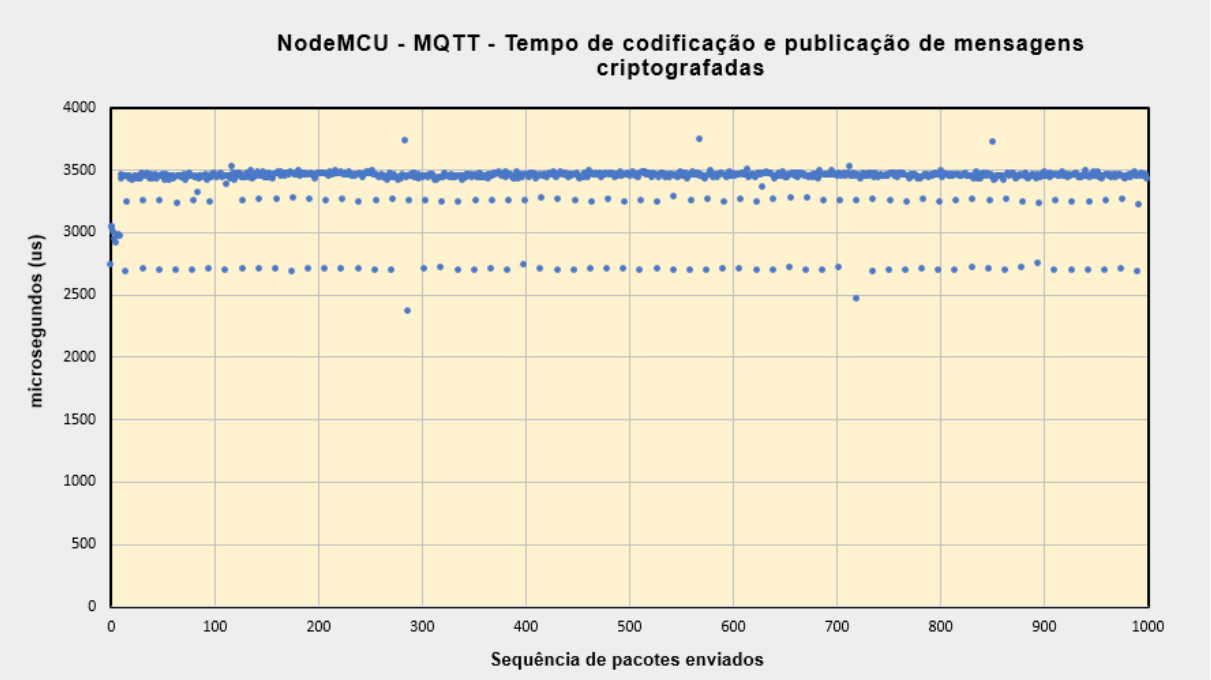
O campo selecionado na Figura 23, *Encrypted Application Data* do protocolo *TLS* está criptografado. Sabe-se que o seu conteúdo é o protocolo *MQTT*, desta forma pode-se concluir que os princípios de *SI* confidencialidade e integridade estão aplicados. Nota-se também que em nenhum momento foi possível identificar qual é o conteúdo do pacote *TLS,* este mesmo pacote, em meio aos milhões de pacotes *TLS* que trafegam pela Internet, contendo diferentes protocolos encapsulados, dificulta a identificação do mesmo.

O tamanho total dos dados da camada de aplicação, soma em *bytes* do tamanho dos protocolos *MQTT* e *TLS* é de 53 *bytes* Comparando com o tamanho

de 18 bytes de um pacote “puro” *MQTT* capturado no cenário 1, identificada uma razão proporcional aproximada de 3 vezes, ou seja, o processo de criptografia, aumenta a quantidade de dados trafegados. Este é o preço computacional pago em relação a obtenção dos princípios de *SI*.

Análise da amostra

Figura 24 - Cenário 2 – Representação visual da amostra gerada



Fonte: próprio autor

O tempo de empacotamento e publicação da mensagem neste cenário foi consideravelmente maior, foram extraídos da análise os seguintes tempos:

Tempo médio: 3389µs

Tempo máximo: 3737µs

Tempo mínimo: 2362µs

Considerando o sensor e suas características o impacto do uso de criptografia ficou evidente que o dispositivo levou em média 3 vezes mais tempo para conseguir processar e enviar os dados em relação ao não uso de criptografia.

Observou-se o mesmo comportamento cíclico do cenário 1, que são os pontos inferiores plotados no gráfico da Figura 24. A razão deste fato explica-se também pelo uso do *buffer* do protocolo *MQTT,* que ao publicar dois pacotes em um

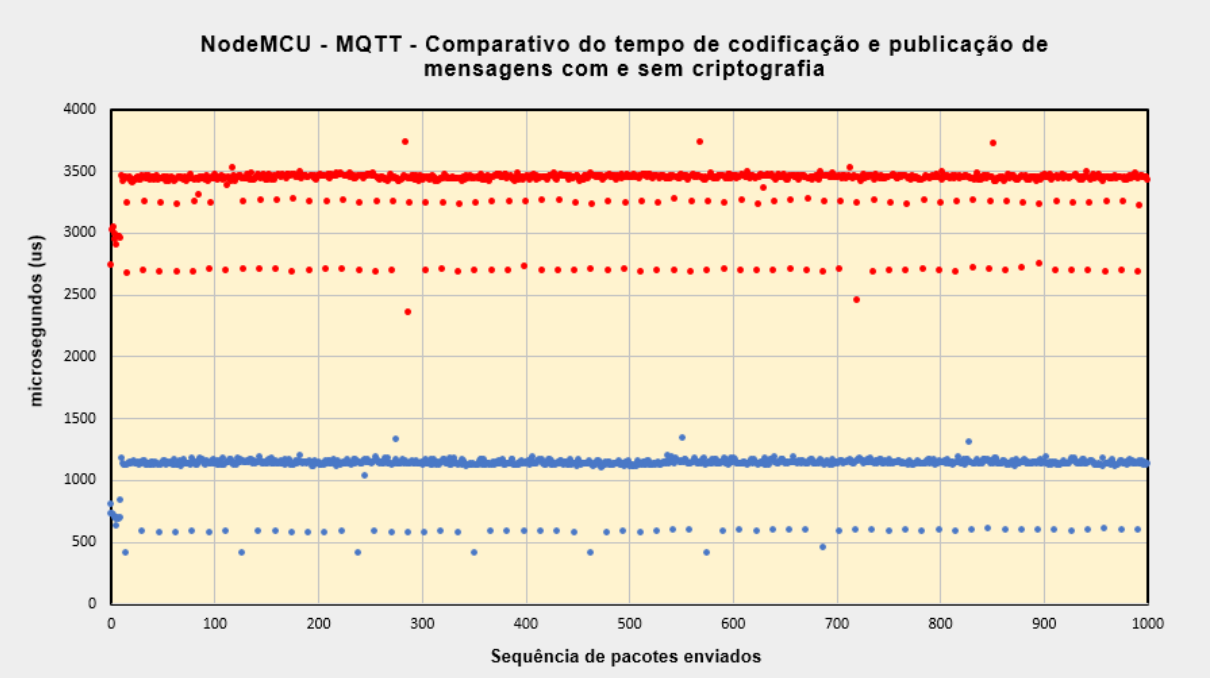
único pacote *TCP*, reduz o tempo de processamento e envio.

Problemas identificados envolvendo segurança da informação

Após a implementação da camada de segurança, ainda permaneceu o problema de disponibilidade. O fato de transmitir os dados criptografados não tem relação a capacidade do equipamento de impedir ataques do padrão *DoS*.

4.5.3 Comparativo entre as duas amostras

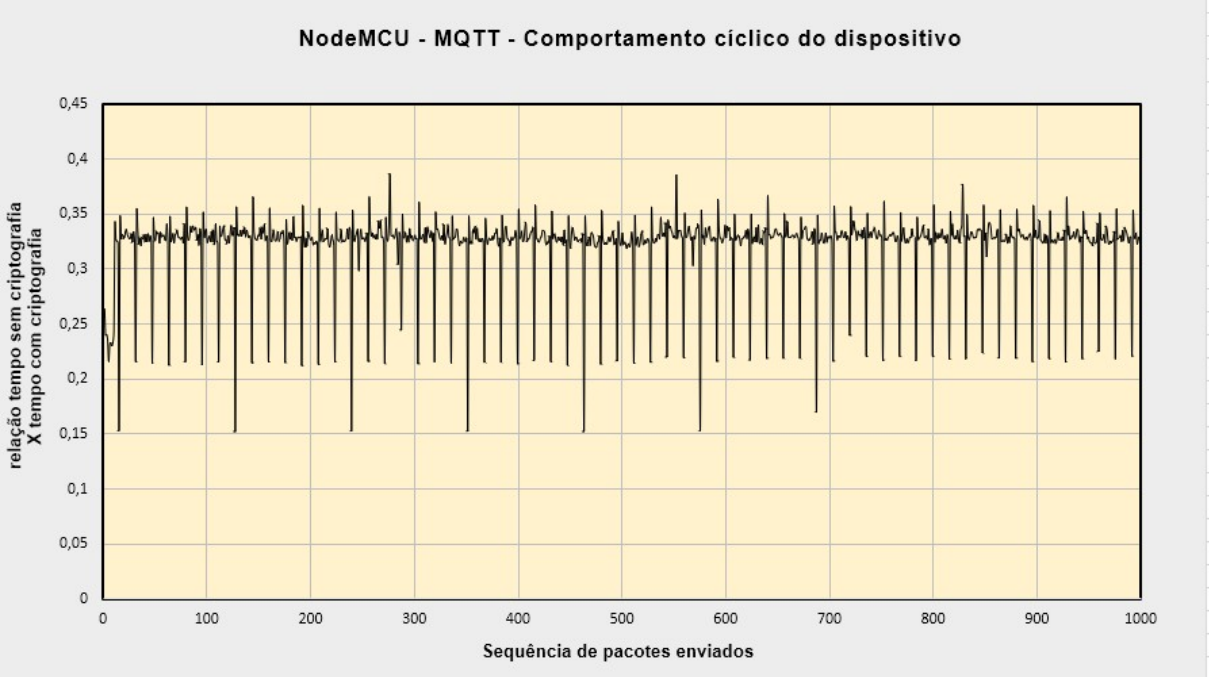
Figura 25 - Comparativo entre as amostras



Fonte: próprio autor

Neste gráfico da Figura 25, as duas amostras foram plotadas, os pontos na cor vermelho são os dados amostrados para o Cenário 2 e os pontos na cor azul são os dados amostrados para o Cenário 1. O sensor se comportou de forma muito similar durante a execução dos dois *softwares* de publicação de mensagens *MQTT.* O gráfico demonstra bem este comportamento, a única diferença foi o tempo maior que o sensor utilizou durante o Cenário 2.

Figura 26 - Relação entre os tempos



Fonte: próprio autor

Somente para expor o comportamento cíclico do sensor durante a geração das duas amostras, os dados resultantes foram agrupados e calculada a razão entre os pontos com mesmo valor de sequência, por exemplo:

Pacote de número sequencial 11

Tempo do Cenário 1: 1169µs

Tempo do Cenário 2: 3424µs

Cálculo (1169 / 3424) = 0,34

Assim observamos que o tempo gasto para processamento e envio de um pacote sem criptografia é em média 30% do tempo gasto pelo mesmo pacote com o uso de criptografia.

5 CONSIDERAÇÕES FINAIS

A Internet das Coisas é um conceito que já se tornou realidade e a cada dia ela está mais presente. Preservar os dados desde o momento de sua coleta, passando pelo processamento local, transmissão, recepção, armazenamento e recuperação é de extrema importância.

Uma das características deste termo é a integração de equipamentos aos recursos de rede, transformando um equipamento comum em um equipamento denominado de “inteligente”, como, por exemplo, uma televisão que até alguns anos atrás só tinha a função de exibir imagens e hoje, uma televisão “inteligente” é capaz de acessar a *Internet* e prover ao usuário recursos adicionais.

Para que este processo de tornar um equipamento comum em um equipamento “inteligente” foi necessário o desenvolvimento de dispositivos com recursos computacionais capazes de possibilitar tal tarefa.

Um dispositivo muito utilizado para prototipagem de soluções *IoT* é o *Arduino*, uma inciativa de *Hardware* livre, onde pode-se comprar os componentes e montar nossa placa bem como comprar uma placa já montada, ele fornece uma interface de desenvolvimento na qual podemos programar, enviar o *software* ao dispositivo e testar a solução. Vários fabricantes utilizam este “ecossistema” já pronto como base para a criação de dispositivos. Um dispositivo baseado nesta plataforma é o *nodeMCU* que foi o dispositivo escolhido para implementar o sensor deste projeto, justamente pela característica de ter recursos computacionais limitados.

Junto com a necessidade da criação de soluções *IoT* um outro fator limitante contribuiu para o surgimento de protocolos específicos a este fim. São conhecidos por protocolos leves que podem ser facilmente implementados em dispositivos de recursos limitados sendo o protocolo escolhido para análise o *MQTT*.

Dois cenários *IoT* foram montados, as amostras foram coletadas, os dados analisados e comparados graficamente.

O principal questionamento abordado por este trabalho foi exatamente responder se um dispositivo popular, considerando vendas e o baixo custo, é capaz de implementar soluções *IoT* de forma a preservar os princípios confidencialidade, integridade e disponibilidade de *SI.*

Defronte a esta questão e aos resultados das análises realizadas foi possível

concluir que o uso do protocolo *MQTT* sem a camada auxiliar de criptografia (discutido no cenário 1) é extremamente vulnerável, a característica leve do protocolo atendeu aos requisitos da rede, mas não foi capaz de proteger os dados trafegados, assim recomenda-se o uso neste formato somente em ambientes de testes onde a leitura dos dados trafegados de forma clara pode responder questões de desempenho da rede ou em ambientes protegidos por outras soluções.

Conclui-se também que ao propor o uso do protocolo *TLS* como uma camada extra de proteção ao protocolo *MQTT* os princípios de *SI,* confidencialidade e integridade foram alcançados, o sensor foi capaz de executar de forma satisfatória os requisitos do protocolo de criptografia *TLS* e considerando a característica de uso de um sensor que é transmitir dados de forma periódica o tempo extra de processamento e publicação de uma mensagem não impactou o seu objetivo. Portanto o *nodeMCU* foi capaz de se comportar como um sensor de forma a garantir os princípios de *SI*.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ATZORI, Luigi; IERA, Antonio; MORABITO, Giacomo. ***The Internet of things: a survey.*** The International Journal of Computer and Telecommunications Networking, Amsterdam. 2010. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128610001568>. Acesso em: 3 jun. 2018

BANKS, Andrew; GUPTA Rahul. ***MQTT Version 3.1.1***. OASIS Standard, 2014. Dispoível em <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Acesso em: 1 mai. 2018.

CARVALHO, M. S. **Conceitos Básicos de Sistemas de Informação Geográfica e Cartografia Aplicados à Saúde**. Brasília: Organização Panamericana de Saúde, Ministério da Saúde, 2000.

CISCO. **Relatório Anual de Cibersegurança 2018**, 2018. Disponível em: <https://www.cisco.com/c/pt_br/products/security/security-reports.html>. Acesso em: 10 abr. 2018.

COMER, D. E. **Interligação em rede com TCP/IP, Vol 1 – Princípios, protocolos e arquitetura**, trad. da 3ª Edição, Campus, 1998.

COULORIS, G.; DOLLIMORE, J; KINDBERG, T. **Sistemas Distribuídos Conceitos e Projeto,** trad. da 4ª Edição, Bookman, 2007.

HIVEMQ MQTT BROKER. ***MQTT Essentials Part 6: Quality of Service 0, 1 & 2***. [s.d.]. Disponível em <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels>. Acesso em: 20 mai. 2018.

IMDA.. ***The Internet of Things (IOT).*** *Info-communications Media Development Authority.* 2017. Disponível em: <https://www.imda.gov.sg/-/media/imda/files/industry-development/infrastructure/technology/internetofthings.pdf?la=en>. Acesso em: 7 abr. 2018.

KUROSE, J. F; ROSS, K. W. ***Computer Networking: A top down approach featuring the Internet*****,** 6ª Edição, Pearson, 2012.

ESPRESSIF. ***ESP8266 Overview***. [s.d.]. Disponível em <https://www.espressif.com/en/products/hardware/esp8266ex/overview>. Acesso em: 10 mai. 2018.

MICHAELIS. ***Dicionário Brasileiro da Língua Portuguesa*** [s.d.]. Editora Melhoramentos Ltda. Disponível em <https://michaelis.uol.com.br>. Acesso em: 15 mai. 2018.

RFC 7452. Request for Comments: 7452. ***Architectural Considerations in Smart Object Networking***, 2015. RFC, 2015. Disponível em <https://tools.ietf.org/html/rfc7452>. Acesso em: 01 jun. 2018.

STALLINGS, W. **Criptografia e segurança de redes: Princípios e práticas.** trad. 4ª Ed. São Paulo: Pearson Prentice Hall, 2008.

TANENBAUM, A. S. **Redes de Computadores.** 4ª Ed. Rio de Janeiro: Elsevier, 2003.

WIRESHARK. ***Wireshark User's Guide Version 2.9.0.*** Disponível em: <https://www.wireshark.org/docs/wsug_html_chunked/>. Acesso em: 1 jun. 2018.

7 ANEXOS

ANEXO A – AMOSTRA DOS DADOS COLETADOS DO CENÁRIO 1

0;723	35;1130	70;1141	105;1122	140;1133
1;801	36;1113	71;1158	106;1155	141;1141
2;729	37;1130	72;1129	107;1117	142;1135
3;719	38;1121	73;1126	108;1170	143;580
4;694	39;1136	74;1125	109;1121	144;1182
5;628	40;1120	75;1130	110;1119	145;1152
6;686	41;1138	76;1133	111;580	146;1134
7;692	42;1120	77;1123	112;1143	147;1136
8;683	43;1145	78;1168	113;1137	148;1130
9;688	44;1125	79;580	114;1139	149;1136
10;839	45;1131	80;1150	115;1157	150;1134
11;1169	46;1121	81;1130	116;1159	151;1156
12;1124	47;576	82;1121	117;1152	152;1134
13;1121	48;1121	83;1135	118;1127	153;1160
14;1116	49;1133	84;1121	119;1121	154;1154
15;409	50;1126	85;1142	120;1131	155;1168
16;1123	51;1148	86;1169	121;1117	156;1126
17;1128	52;1146	87;1158	122;1143	157;1132
18;1140	53;1136	88;1156	123;1177	158;1134
19;1132	54;1118	89;1162	124;1143	159;583
20;1125	55;1141	90;1125	125;1145	160;1151
21;1136	56;1126	91;1134	126;1158	161;1133
22;1150	57;1133	92;1114	127;410	162;1128
23;1136	58;1138	93;1150	128;1150	163;1145
24;1133	59;1128	94;1137	129;1137	164;1156
25;1126	60;1122	95;576	130;1139	165;1134
26;1132	61;1167	96;1133	131;1145	166;1128
27;1137	62;1115	97;1132	132;1154	167;1134
28;1116	63;572	98;1155	133;1137	168;1158
29;1144	64;1115	99;1157	134;1151	169;1120
30;1124	65;1122	100;1126	135;1157	170;1133
31;582;	66;1129	101;1129	136;1159	171;1155
32;1147;	67;1146	102;1135	137;1130	172;1149
33;1155;	68;1112	103;1117	138;1151	...
34;1117	69;1138	104;1150	139;1141	1000;1128

ANEXO B – AMOSTRA DOS DADOS COLETADOS DO CENÁRIO 2

0;2737	35;3447	70;3435	105;3456	140;3453
1;3030	36;3468	71;3457	106;3447	141;3450
2;3043	37;3443	72;3430	107;3466	142;3476
3;2994	38;3430	73;3417	108;3435	143;2704
4;2947	39;3438	74;3440	109;3451	144;3262
5;2910	40;3456	75;3453	110;3447	145;3445
6;2968	41;3429	76;3447	111;2694	146;3461
7;2971	42;3449	77;3472	112;3384	147;3475
8;2975	43;3447	78;3443	113;3451	148;3467
9;2963	44;3460	79;2692	114;3443	149;3435
10;3463	45;3427	80;3254	115;3447	150;3434
11;3424	46;3444	81;3432	116;3424	151;3436
12;3449	47;2688	82;3445	117;3529	152;3471
13;3444	48;3250	83;3438	118;3451	153;3437
14;3449	49;3443	84;3313	119;3468	154;3463
15;2680	50;3455	85;3426	120;3419	155;3432
16;3242	51;3439	86;3451	121;3456	156;3458
17;3442	52;3456	87;3438	122;3450	157;3429
18;3448	53;3418	88;3450	123;3467	158;3461
19;3423	54;3457	89;3442	124;3454	159;2704
20;3432	55;3427	90;3444	125;3464	160;3265
21;3410	56;3417	91;3432	126;3438	161;3466
22;3424	57;3444	92;3459	127;2704	162;3477
23;3430	58;3441	93;3419	128;3251	163;3464
24;3444	59;3451	94;3433	129;3438	164;3452
25;3441	60;3445	95;2706	130;3440	165;3475
26;3451	61;3423	96;3243	131;3470	166;3467
27;3429	62;3431	97;3441	132;3444	167;3469
28;3441	63;2692	98;3456	133;3454	168;3460
29;3426	64;3229	99;3445	134;3431	169;3442
30;3464	65;3451	100;3474	135;3489	170;3472
31;2700	66;3450	101;3452	136;3434	171;3473
32;3251	67;3451	102;3443	137;3429	172;3467
33;3443	68;3438	103;3431	138;3433	…
34;3459	69;3444	104;3443	139;3442	1000;3430

ANEXO C – TCC_NODEMCU_C01.INO (LISTAGEM DO *SOFTWARE*)

```
/*  FATEC Americana 2018
    Curso de SeguranÃ§a da InformaÃ§Ã£o
    Trabalho final de conclusÃ£o de curso
    Marcos FlÃ¡vio Eli Periera

    NodeMCU - ImplementaÃ§Ã£o de sensor IOT para trasmissÃ£o da temperatura ambiente (simulada)
    via MQTT e disponibilizaÃ§Ã£o de acesso via telnet para monitoramento do sistema.

    versÃ£o 0.0.2 24/05/2018

*/

#include <NTPClient.h>
#include <WiFiUdp.h>
#include <WiFiClientSecure.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <time.h>                       // time() ctime()

#define MQTT_SUB1 "IOT/CMD"
#define MQTT_PUB1 "IOT/MSG"
#define MQTT_ID "MCU1979"

#define MQTT_PORT 1883
#define MQTT_IPV4 "192.168.1.51"

#define WIFI_SSID "IOT"
#define WIFI_PASS "28417300856"

#define DEBUG_SERIAL true   // Permite o monitoramento via porta serial USB

#define MAX_SRV_CLIENTS 3
#define TELNET_BANNER "\n\r:: NODEMCU IOT TELNET SERVER :: "
#define TELNET_OPCOES "\n\r:: OpÃ§Ãµes [E]pochTime [U]ptime [S]air \n\r:: OpÃ§Ã£o: "


//globais

WiFiServer server(23);
WiFiClient wc[MAX_SRV_CLIENTS];

//WiFiClient wc;
PubSubClient mqtt(wc[0]); // Instancia o Cliente MQTT passando o objeto wc

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "192.168.1.1", -10800, 60000);

int led_ligado; //Estado do LED embutido na placa do NodeMCU 1 = Ligado e 0 = Desligado
char espera;

char strbuffer[40];

uint32_t now_ms, now_us;

int seq;

//pragma
void mqtt_rcvmsg(char* topic, byte* payload, unsigned int length);
void led(int liga);
int led_estado();

uint8_t i = 0;
uint8_t j = 0;


//------------------------------------------------------------SETUP
void setup() {

  seq = 0;
```

```
  espera = '\\';

  // put your setup code here, to run once:
  pinMode(2, OUTPUT); //LED embutido na placa

  //Monitoramento Serial
  if (DEBUG_SERIAL) Serial.begin(115200);

  //ConexÃ£o WiFi
  wifi_conecta();

  //ConexÃ£o com BROKER MQTT
  mqtt.setServer(MQTT_IPV4, MQTT_PORT);
  mqtt.setCallback(mqtt_rcvmsg);

  if (mqtt.connect(MQTT_ID)) {
    dsprint("\nConectado com sucesso ao broker MQTT!");
    mqtt.subscribe(MQTT_SUB1);
  }

  //Inicia o servidor TELNET
  server.begin();
  server.setNoDelay(true);


  timeClient.begin();
  timeClient.update();

}


//-----------------------------------------------------------------LOOP
void loop() {

  //-------------------------------------------------------------------

  if (!mqtt.loop()) {
    wifi_conecta();
    dsprint("\nConectando ao broker MQTT\n");
    while (!mqtt.connect(MQTT_ID)) {
      dswaiting();
      delay(200);
    }
    dsprint("\r--> Conectado! \n\r--> InscriÃ§Ã£o no tÃ³pico: "); dsprint(MQTT_SUB1);
    if (mqtt.subscribe(MQTT_SUB1), 1) {
      dsprint(" OK");
    } else {
      dsprint(" Falha!");
    }
  }

  //-------------------------------------------------------------------TELNET SERVER

  //dsprint("\nTELNET: Rotinas...");

  //check if there are any new clients
  if (server.hasClient()) {
    dsprint("\nTELNET: Verificando por novos clientes...");
    for (i = 1; i < MAX_SRV_CLIENTS; i++) {
      //find free/disconnected spot
      if (!wc[i] || !wc[i].connected()) {
        if (wc[i]) wc[i].stop();
        wc[i] = server.available();
        dsprint("\nTELNET: Novo cliente.: "); dsprint(i);

        wc[i].write(TELNET_BANNER); wc[i].print(timeClient.getFormattedTime());
wc[i].write(TELNET_OPCOES);

        delay(10);

        wc[i].flush();

        break;
      }
```

```
    }
    //no free/disconnected spot so reject
    if ( i == MAX_SRV_CLIENTS) {
      WiFiClient serverClient = server.available();
      serverClient.stop();
      dsprint("\nTELNET: ConexÃ£o rejeitada! ");
    }
  }


  //check clients for data
  for (i = 1; i < MAX_SRV_CLIENTS; i++) {
    if (wc[i] && wc[i].connected()) {
      if (wc[i].available()) {
        //get data from the telnet client and push it to the UART
        dsprint("\nTELNET ["); dsprint(i); dsprint("]: ");

        size_t len = wc[i].available();
        uint8_t sbuf[len];
        wc[i].readBytes(sbuf, len);
        if (DEBUG_SERIAL) Serial.write(sbuf, len);

        switch (sbuf[0]) {
          case 'U':
            now_ms = millis();
            wc[i].write("\n\r:: Uptime (ms) "); wc[i].print(now_ms);
            break;
          case 'E':
            wc[i].write("\n\r:: ");
            wc[i].print(timeClient.getEpochTime());
            break;
          case 'S':
            wc[i].stop();
            break;
          default:
            wc[i].write("\n\r:: nÃ£o implementado!");
        }
        wc[i].write(TELNET_BANNER); wc[i].print(timeClient.getFormattedTime());
wc[i].write(TELNET_OPCOES);
        wc[i].write(TELNET_OPCOES);

      }
    }
  }

  //check UART for data
  if (Serial.available()) {
    size_t len = Serial.available();
    uint8_t sbuf[len];
    Serial.readBytes(sbuf, len);
    //push UART data to all connected telnet clients
    for (i = 1; i < MAX_SRV_CLIENTS; i++) {
      if (wc[i] && wc[i].connected()) {
        wc[i].write(sbuf, len);
        delay(1);
      }
    }
  }



  //--------------------------------------------------------------------------------

  if (seq <= 1000) {

    sprintf(strbuffer, "MCU Monitor seq: %d", seq);

    now_us = micros();
    mqtt.publish(MQTT_PUB1, strbuffer, 1);
    now_us = micros() - now_us;

    sprintf(strbuffer, "%d;%d", seq, now_us);
    mqtt.publish(MQTT_PUB1, strbuffer, 1);
```

```
    delay(1000);
    seq++;

  }

  //--------------------------------------------------------------------------------

}


void dsprint(String msg) {
  if (DEBUG_SERIAL) Serial.print(msg);
}

void dsprint(int msg) {
  if (DEBUG_SERIAL) Serial.print(msg);
}

void dswaiting() {
  if (DEBUG_SERIAL) {
    Serial.write("\b");
    Serial.write(espera);
  }
  switch (espera) {
    case '-': espera = '\\'; break;
    case '\\': espera = '|'; break;
    case '|': espera = '/'; break;
    case '/': espera = '-'; break;
  }
}

bool wifi_conecta() {
  if (WiFi.status() != WL_CONNECTED) {
    dsprint("\n\rIniciando a conexÃ£o WiFi ("); dsprint(WIFI_SSID); dsprint(")\n\r");
    WiFi.begin(WIFI_SSID, WIFI_PASS); // Conecta na rede WI-FI
    while (WiFi.status() != WL_CONNECTED) {
      delay(200);
      dswaiting();
    }
    dsprint("\r --> Conectado!");
  }
}

void led(int liga) {
  if (liga == 0) digitalWrite(2, HIGH);   // Acende o Led
  else digitalWrite(2, LOW);    // Apaga o Led
  dsprint("\n- FunÃ§Ã£o led parÂ¢metro liga = ");
  dsprint(liga);
}

int led_estado() {
  led_ligado = digitalRead(2);
  return led_ligado;
}

void mqtt_rcvmsg(char* topic, byte* payload, unsigned int length) {

  String msg;

  //obtem a string do payload recebido
  for (int i = 0; i < length; i++) {
    char c = (char) payload[i];
    msg += c;
  }

  dsprint("\nMQTT Message Incoming <== ");
  dsprint(msg);

  if (msg.equals("MCU LED LIGA")) {
    dsprint("IF LIGA");
    led(1);
  }

  if (msg.equals("MCU LED DESLIGA")) {
```

```
    dsprint("IF DESLIGA");
    led(0);
  }

}
```

ANEXO D – TCC_NODEMCU_C02.INO (LISTAGEM DO *SOFTWARE*)

```
/*  FATEC Americana 2018
    Curso de SeguranÃ§a da InformaÃ§Ã£o
    Trabalho final de conclusÃ£o de curso
    Marcos FlÃ¡vio Eli Periera

    NodeMCU - ImplementaÃ§Ã£o de sensor IOT para trasmissÃ£o da temperatura ambiente (simulada)
    via MQTT e disponibilizaÃ§Ã£o de acesso via telnet para monitoramento do sistema.

    versÃ£o 0.0.2 24/05/2018

*/


#include <NTPClient.h>
#include <WiFiUdp.h>
#include <WiFiClientSecure.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <time.h>                       // time() ctime()

#include "FS.h"

#define MQTT_SUB1 "IOT/CMD"
#define MQTT_PUB1 "IOT/MSG"
#define MQTT_ID "MCU1979"

#define MQTT_PORT 1883
#define MQTT_IPV4 "192.168.1.51"

#define WIFI_SSID "IOT"
#define WIFI_PASS "28417300856"

#define DEBUG_SERIAL true   // Permite o monitoramento via porta serial USB

#define MAX_SRV_CLIENTS 3
#define TELNET_BANNER "\n\r:: NODEMCU IOT TELNET SERVER :: "
#define TELNET_OPCOES "\n\r:: OpÃ§Ãµes [E]pochTime [U]ptime [S]air \n\r:: OpÃ§Ã£o: "

//pragma
void mqtt_rcvmsg(char* topic, byte* payload, unsigned int length);
void led(int liga);
int led_estado();

//globais
WiFiServer server(23);
WiFiClient wc[MAX_SRV_CLIENTS];
WiFiClientSecure wcsec;

PubSubClient mqtt("broker.lan", 8883, mqtt_rcvmsg, wcsec);

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "192.168.1.1", -10800, 60000);

int led_ligado; //Estado do LED embutido na placa do NodeMCU 1 = Ligado e 0 = Desligado
char espera;

char strbuffer[40];

uint32_t now_ms, now_us;

int seq;

uint8_t i = 0;
uint8_t j = 0;


//------------------------------------------------------------SETUP
void setup() {
```

```
  seq = 0;

  espera = '\\';

  // put your setup code here, to run once:
  pinMode(2, OUTPUT); //LED embutido na placa

  //Monitoramento Serial
  if (DEBUG_SERIAL) Serial.begin(115200);

  SPIFFS.begin();
  File ca = SPIFFS.open("/mcu.lan.crt.der", "r");
  if (!ca) {
    dsprint("\nNao foi possivel acessar o arquivo /mcu.lan.crt.der");
    return;
  }

  if (wcsec.loadCertificate(ca)) {
    dsprint("\nCertificado mcu.lan.crt.der OK");
  } else {
    dsprint("\nNao foi possivel carregar o certificado mcu.lan.crt.der");
    return;
  }

  File key = SPIFFS.open("/mcu.lan.key.der", "r");
  if (!key) {
    Serial.println("\nCouldn't load key");
    return;
  }

  if (wcsec.loadPrivateKey(key)) {
    Serial.println("\nChave mcu.lan.key.der OK");
  } else {
    dsprint("\nNao foi possivel carregar a chave mcu.lan.key.der");
  }

  //ConexÃ£o WiFi
  wifi_conecta();

  //ConexÃ£o com BROKER MQTT
  //mqtt.setServer(MQTT_IPV4, MQTT_PORT);
  //mqtt.setCallback(mqtt_rcvmsg);

  if (mqtt.connect(MQTT_ID,"mcu","fatec")) {
    dsprint("\nConectado com sucesso ao broker MQTT!");
    mqtt.subscribe(MQTT_SUB1);
  }

  //Inicia o servidor TELNET
  server.begin();
  server.setNoDelay(true);


  timeClient.begin();
  timeClient.update();

}


//---------------------------------------------------------------LOOP
void loop() {

  //-----------------------------------------------------------------

  if (!mqtt.loop()) {
    wifi_conecta();
    dsprint("\nConectando ao broker MQTT\n");
    while (!mqtt.connect(MQTT_ID,"mcu","fatec")) {
      dswaiting();
      delay(1000);
    }
    dsprint("\r--> Conectado! \n\r--> InscriÃ§Ã£o no tÃ³pico: "); dsprint(MQTT_SUB1);
    if (mqtt.subscribe(MQTT_SUB1), 1) {
```

```
      dsprint(" OK");
    } else {
      dsprint(" Falha!");
    }
  }

  //------------------------------------------------------------------------TELNET SERVER

  //dsprint("\nTELNET: Rotinas...");

  //check if there are any new clients
  if (server.hasClient()) {
    dsprint("\nTELNET: Verificando por novos clientes...");
    for (i = 1; i < MAX_SRV_CLIENTS; i++) {
      //find free/disconnected spot
      if (!wc[i] || !wc[i].connected()) {
        if (wc[i]) wc[i].stop();
        wc[i] = server.available();
        dsprint("\nTELNET: Novo cliente.: "); dsprint(i);

        wc[i].write(TELNET_BANNER); wc[i].print(timeClient.getFormattedTime());
wc[i].write(TELNET_OPCOES);

        delay(10);

        wc[i].flush();

        break;
      }
    }
    //no free/disconnected spot so reject
    if ( i == MAX_SRV_CLIENTS) {
      WiFiClient serverClient = server.available();
      serverClient.stop();
      dsprint("\nTELNET: ConexÃ£o rejeitada! ");
    }
  }


  //check clients for data
  for (i = 1; i < MAX_SRV_CLIENTS; i++) {
    if (wc[i] && wc[i].connected()) {
      if (wc[i].available()) {
        //get data from the telnet client and push it to the UART
        dsprint("\nTELNET ["); dsprint(i); dsprint("]: ");

        size_t len = wc[i].available();
        uint8_t sbuf[len];
        wc[i].readBytes(sbuf, len);
        if (DEBUG_SERIAL) Serial.write(sbuf, len);

        switch (sbuf[0]) {
          case 'U':
            now_ms = millis();
            wc[i].write("\n\r:: Uptime (ms) "); wc[i].print(now_ms);
            break;
          case 'E':
            wc[i].write("\n\r:: ");
            wc[i].print(timeClient.getEpochTime());
            break;
          case 'S':
            wc[i].stop();
            break;
          default:
            wc[i].write("\n\r:: nÃ£o implementado!");
        }
        wc[i].write(TELNET_BANNER); wc[i].print(timeClient.getFormattedTime());
wc[i].write(TELNET_OPCOES);
        wc[i].write(TELNET_OPCOES);

      }
    }
  }
```

```
  //check UART for data
  if (Serial.available()) {
    size_t len = Serial.available();
    uint8_t sbuf[len];
    Serial.readBytes(sbuf, len);
    //push UART data to all connected telnet clients
    for (i = 1; i < MAX_SRV_CLIENTS; i++) {
      if (wc[i] && wc[i].connected()) {
        wc[i].write(sbuf, len);
        delay(1);
      }
    }
  }



  //--------------------------------------------------------------------------------

  if (seq <= 1000) {

    sprintf(strbuffer, "MCU Monitor seq: %d", seq);

    now_us = micros();
    mqtt.publish(MQTT_PUB1, strbuffer, 1);
    now_us = micros() - now_us;

    sprintf(strbuffer, "%d;%d", seq, now_us);
    mqtt.publish(MQTT_PUB1, strbuffer, 1);

    delay(1000);
    seq++;

  }

  //--------------------------------------------------------------------------------

}


void dsprint(String msg) {
  if (DEBUG_SERIAL) Serial.print(msg);
}

void dsprint(int msg) {
  if (DEBUG_SERIAL) Serial.print(msg);
}

void dswaiting() {
  if (DEBUG_SERIAL) {
    Serial.write("\b");
    Serial.write(espera);
  }
  switch (espera) {
    case '-': espera = '\\'; break;
    case '\\': espera = '|'; break;
    case '|': espera = '/'; break;
    case '/': espera = '-'; break;
  }
}

bool wifi_conecta() {
  if (WiFi.status() != WL_CONNECTED) {
    dsprint("\n\rIniciando a conexÃ£o WiFi ("); dsprint(WIFI_SSID); dsprint(")\n\r");
    WiFi.begin(WIFI_SSID, WIFI_PASS); // Conecta na rede WI-FI
    while (WiFi.status() != WL_CONNECTED) {
      delay(200);
      dswaiting();
    }
    dsprint("\r --> Conectado!");
  }
}

void led(int liga) {
  if (liga == 0) digitalWrite(2, HIGH);   // Acende o Led
```

```
  else digitalWrite(2, LOW);    // Apaga o Led
  dsprint("\n- FunÃ§Ã£o led parÃ¢metro liga = ");
  dsprint(liga);
}

int led_estado() {
  led_ligado = digitalRead(2);
  return led_ligado;
}

void mqtt_rcvmsg(char* topic, byte* payload, unsigned int length) {

  String msg;

  //obtem a string do payload recebido
  for (int i = 0; i < length; i++) {
    char c = (char) payload[i];
    msg += c;
  }

  dsprint("\nMQTT Message Incoming <== ");
  dsprint(msg);

  if (msg.equals("MCU LED LIGA")) {
    dsprint("IF LIGA");
    led(1);
  }

  if (msg.equals("MCU LED DESLIGA")) {
    dsprint("IF DESLIGA");
    led(0);
  }

}
```

ANEXO E – *SCRIPTS LINUX* UTILIZADOS

Arquivo coleta_cenario01.sh

```
#!/bin/sh
mosquitto_sub -h 192.168.1.51 -t "IOT/MSG" | grep -v -w "MCU"
```

Arquivo coleta_cenario02.sh

```
#!/bin/sh
mosquitto_sub -h broker.lan -p 8883 -q 1 -t IOT/MSG \
 --cafile /etc/mosquitto/certs/broker.lan.crt.pem \
--tls-version tlsv1.1 -u broker -P fatec -i mosqsub/broker \
 | grep -v -w "MCU"
```

ANEXO F – PROCESSO DE CRIAÇÃO DOS CERTIFICADOS DIGITAIS

Os certificados são criados utilizando o *software* OpenSSL e conforme os passos descritos abaixo:

A) Criar o certificado CA.

openssl req -new -x509 -days 3650 -extensions v3_ca -keyout ca.key.pem -out ca.crt.pem

B) Criar as chaves pública e privada para o *broker*, o certificado de requisição de assinatura e o certificado assinado, respectivamente.

openssl genrsa -out broker.lan.key.pem 2048
openssl req -out broker.lan.csr.pem -key broker.lan.key.pem -new
openssl x509 -req -in broker.lan.csr.pem -CA ca.crt.pem -CAkey ca.key.pem -CAcreateserial -out broker.lan.crt.pem -days 365

Ao final deste passo estamos com os certificados prontos para uso no arquivo de configuração do *broker* do *mosquitto*.

C) Criar as chaves pública e privada para o cliente *MQTT*, o certificado de requisição de assinatura e o certificado assinado, respectivamente.

openssl genrsa -out mcu.lan.key.pem 2048
openssl req -out mcu.lan.csr.pem -key mcu.lan.key.pem -new
openssl x509 -req -in mcu.lan.csr.pem -CA ca.crt.pem -CAkey ca.key.pem -CAcreateserial -out mcu.lan.crt.pem -days 365

Para possibiliar a leitura dos certificados pelo dispositivo *nodeMCU* é necessário converter do formato PEM (formato *ASCII*) para o formato DER que é o formato binário do certificado.

openssl x509 -outform der -in mcu.lan.crt.pem -out mcu.lan.crt.der

openssl rsa -outform der -in mcu.lan.key.pem -out mcu.lan.key.der

Comando extra não utilizado.

openssl x509 -in mosquitto/etc/certs/broker.lan.crt.pem -sha1 -noout -fingerprint