

Johann Westphall

**Desenvolvimento de uma Aplicação com
dispositivo IoT usando Protocolos DTLS e
CoAP**

Brasil

2018

Johann Westphall

Desenvolvimento de uma Aplicação com dispositivo IoT usando Protocolos DTLS e CoAP

Trabalho de Conclusão de Curso submetido
ao Curso de Ciências da Computação para a
obtenção do Grau de Bacharel em Ciências
da Computação.

Universidade Federal de Santa Catarina
Centro Tecnológico - CTC
Departamento de Informática e Estatística
Ciências da Computação

Orientador: Carla Merkle Westphall
Coorientador: Bel. Leandro Loffi

Brasil
2018

Johann Westphall

Desenvolvimento de uma Aplicação com dispositivo IoT usando Protocolos DTLS e CoAP/ Johann Westphall. – Brasil, 2018-
83 p. : il. (algumas color.) ; 30 cm.

Orientador: Carla Merkle Westphall

Dissertação (Bacharelado) – Universidade Federal de Santa Catarina
Centro Tecnológico - CTC
Departamento de Informática e Estatística
Ciências da Computação, 2018.

1. Internet of Things. 2. Segurança em IoT. I. Carla Merkle Westphall. II. Universidade Federal de Santa Catarina. III. Bacharelado em Ciências da Computação. IV. Dispositivos limitados presentes em IoT, seus protocolos e seus meios de segurança

CDU 02:141:005.7

Johann Westphall

Desenvolvimento de uma Aplicação com dispositivo IoT usando Protocolos DTLS e CoAP

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de Bacharel em Ciências da Computação, e aprovado em sua forma final pelo Curso de Ciências da Computação da Universidade Federal de Santa Catarina.

Profa. Dra. **Carla Merkle Westphall**
Orientador

Bel. **Leandro Loffi**
Co-orientador

Prof. Dr. **Jean Everson Martina**
Membro da banca

Dr. **Jorge Werner**
Membro da banca

Brasil
2018

Agradecimentos

Agradeço inicialmente aos meus pais, que sempre me deram todo o suporte necessário durante minha vida, aos meus orientadores, que providenciaram direção e suporte ao longo deste trabalho, aos meus professores, pela dedicação empregada e o conhecimento que me foi transmitido, aos meus colegas de curso, que me ajudaram e me deram a oportunidade de exercitar meu aprendizado e aos meus amigos, que também fizeram parte desta jornada.

Agradeço em especial a Leandro Loffi, mestrando em Ciências da Computação na UFSC e co-orientador deste trabalho, por ter providenciado grande parte do hardware utilizado no desenvolvimento deste trabalho, por providenciar modelos para o desenvolvimento da parte textual e sugerir bons caminhos para a implementação realizada neste trabalho, quando dificuldades foram encontradas.

Resumo

No contexto de Internet das Coisas, o uso crescente de sensores e equipamentos interligados através da Internet junto ao fato de muitos dispositivos terem limitações energéticas, limitações de processamento e limitações de memória, gera uma demanda por soluções eficientes e compatíveis com a Internet.

Assim, certos protocolos foram criados para facilitar a comunicação de dispositivos limitados com a Internet, com o objetivo de não sobrecarregá-los. CoAP (*Constrained Application Protocol*) e DTLS (*Datagram Transport Layer Security*) são exemplos de protocolos voltados a dispositivos restritos e representam, respectivamente, protocolo de aplicação e protocolo de segurança.

Para resolver um problema da atividade agrícola, envolvendo sensores e um dispositivo de IoT (*Internet of Things*), foi realizado um estudo, em forma de revisão bibliográfica, acerca de segurança em IoT, identificando métodos e protocolos para a transmissão dos dados, coletados por sensores, seguramente. A partir do conhecimento desses métodos e protocolos, foi desenvolvido um protótipo para a solução do problema agrícola.

Palavras-chaves: Internet das Coisas. segurança. limitações. compatibilidade.

Abstract

When dealing with Internet of Things, due to the rising popularity of sensing and small computing devices use through Internet and considering their power, processing and storage limitations, there is a need of efficient and compatible solutions regarding these kind of devices and technology.

Some specific protocols were developed aiming to help the communication of restricted small devices, through the Internet, without overloading them. CoAP (Constrained Application Protocol) and DTLS (Datagram Transport Layer Security) represent, respectively, application protocol and security protocol and both were developed to be used by restricted devices.

In order to solve a problem regarding an agricultural production, that can be solved with sensors and an IoT device, a literature review about IoT security was made, covering methods and protocols to obtain secure data transmission. By the time the study was complete, a prototype to represent the agricultural problem was developed in order to achieve the solution.

Key-words: Internet of Things. security. restrictions. compatibility.

Lista de ilustrações

Figura 1 – Elementos medidos por sensores	24
Figura 2 – Topologia de uma rede com 6LoWPAN	25
Figura 3 – Compressão em relação ao IPv6	26
Figura 4 – Pilha de protocolos	26
Figura 5 – GET HTTP	28
Figura 6 – Cabeçalho CoAP	29
Figura 7 – Compatibilidade entre CoAP e HTTP via proxy	31
Figura 8 – Registro DTLS	34
Figura 9 – Autenticação Completa DTLS	36
Figura 10 – Chip proposto	40
Figura 11 – Sensor de pH e temperatura conectados ao controlador	41
Figura 12 – Sensor de fluxo e ESP8266 conectados	42
Figura 13 – Uso de IoT em aplicação agrícola	43
Figura 14 – DS18B20	45
Figura 15 – Sensor de fluxo	45
Figura 16 – Nodemcu	46
Figura 17 – Pinos físicos	47
Figura 18 – Raspberry Pi	47
Figura 19 – Cipher Suite resultante da atribuição ciphers=“RSA” no código fonte	53
Figura 20 – Captura <i>handshake</i> DTLS (Client Hello)	55
Figura 21 – Captura <i>handshake</i> DTLS (Server Hello)	55
Figura 22 – Interação entre cliente e servidor utilizando CoAP e DTLS	57
Figura 23 – Interação entre Raspberry Pi e servidor utilizando CoAP e DTLS	58

Lista de abreviaturas e siglas

TCP	Transmission Control Protocol
IoT	Internet of Things
RSA	Rivest-Shamir-Adleman
AES	Advanced Encryption Standard
CoAP	Constrained Application Protocol
HTTP	HyperText Transfer Protocol
DTLS	Datagram Transport Layer Security
TLS	Transport Layer Security
WSN	Wireless Sensor Networks
RFC	Request For Comments
GPS	Global Positioning System
PWM	Pulse Width Modulation
UART	Universal asynchronous receiver/transmitter
SPI	Serial Peripheral Interface

Sumário

1	INTRODUÇÃO E OBJETIVOS	19
1.1	Introdução	19
1.2	Problematização	19
1.2.1	Solução Proposta	19
1.3	Metodologia	20
1.3.1	Revisão bibliográfica	20
1.3.2	Entendimento de aplicação dos conceitos	20
1.3.3	Implementação	20
1.3.4	Testes	20
1.3.5	Análise	20
1.4	Justificativa	21
1.5	Objetivo geral	21
1.6	Objetivos específicos	21
1.7	Estrutura do documento	21
2	INTERNET OF THINGS	23
2.1	Aplicações	23
2.2	Limitações	24
2.3	Protocolos para IoT	24
2.4	6LoWPAN - Low-Power Wireless Personal Area Networks	25
2.5	CoAP - Constrained Application Protocol	27
2.5.1	Cabeçalho	29
2.5.1.1	Version	29
2.5.1.2	Type	29
2.5.1.3	Token Length	29
2.5.1.4	Code	29
2.5.1.5	Message ID	30
2.5.1.6	Token	30
2.5.1.7	Options	30
2.5.1.8	Payload	30
2.5.2	Integração com HTTP	30
2.5.3	Transmissão de dados grandes	31
2.5.4	Função Observador	31
2.5.5	Segurança	32
3	SEGURANÇA EM IOT	33

3.1	Segurança a partir de gateway	33
3.2	DTLS - Datagram Transport Layer Security	33
3.2.1	Registro do DTLS	34
3.2.1.1	Tipo de Conteúdo da Mensagem	34
3.2.1.2	Versão do DTLS	34
3.2.1.3	Número de sequência	34
3.2.1.4	Época da mensagem	35
3.2.1.5	Vetor de Inicialização	35
3.2.1.6	Payload	35
3.2.1.7	MAC	35
3.2.2	Handshake	35
3.2.3	Cipher Suite	37
3.2.3.1	Algoritmo de troca de chaves	37
3.2.3.2	Algoritmo de autenticação	37
3.2.3.3	Algoritmo de cifra simétrica	37
3.2.3.4	Algoritmo de HMAC	37
4	TRABALHOS CORRELATOS	39
4.1	Trabalho 1: Constrained Application Protocol (CoAP) no Arduino UNO R3: Uma Análise Prática	39
4.2	Trabalho 2: An energy-efficient reconfigurable DTLS cryptographic engine for End-to-End security in iot applications	39
4.3	Trabalho 3: Water quality monitoring with internet of things (IoT)	40
4.4	Trabalho 4: Telemetry for domestic water consumption based on IoT and open standards	41
4.5	Comparação com o trabalho presente	42
5	DESENVOLVIMENTO DE UMA APLICAÇÃO COM DISPOSITIVOS IOT USANDO DTLS E COAP	43
5.1	Ambiente usado como prova de conceito	43
5.2	Hardware usado	44
5.2.1	Sensor de temperatura	44
5.2.2	Sensor de fluxo	45
5.2.3	NodeMCU	46
5.2.4	Raspberry Pi 3	47
5.3	Bibliotecas	48
5.3.1	CoAP: biblioteca ESP-CoAP	48
5.3.2	CoAP e DTLS: biblioteca CoAPthon	49
5.4	Implementação e testes	49
5.4.1	Desenvolvimento da parte servidor	50

5.4.2	Desenvolvimento da parte cliente	52
5.4.2.1	Cliente CoAP com DTLS	52
5.4.2.2	Código principal do cliente	55
5.4.3	Mensagens de Debug	56
5.4.4	Testes	56
5.4.4.1	Teste inicial	57
5.4.4.2	Teste com Raspberry Pi e sensor de temperatura	57
6	CONCLUSÃO	61
6.1	Principais Contribuições	61
6.2	Sugestões de trabalhos futuros	61
	REFERÊNCIAS	63
	APÊNDICES	67
	APÊNDICE A – CÓDIGO DESENVOLVIDO	69
	APÊNDICE B – ARTIGO NO FORMATO SBC	83

1 Introdução e objetivos

1.1 Introdução

O desenvolvimento computacional permitiu a criação de menores transistores e, por consequência, computadores cada vez menores e mais eficientes. Essa evolução facilitou a diversificação do uso de computadores, como é o caso de sensores e microcontroladores. Todavia, sensores e microcontroladores possuem limitações de alimentação energética, processamento e armazenamento, exigindo protocolos eficientes e eficazes (Granjal; Monteiro; Silva, 2015).

Assim como computadores de uso pessoal, dispositivos pequenos, como sensores e microcontroladores, também transmitem mensagens de valor sigiloso e portanto necessitam de suporte a segurança em sua comunicação. Então, esse trabalho abordará protocolos específicos para dispositivos limitados que fazem parte da Internet das Coisas.

1.2 Problematização

No contexto da agricultura, produtores fazem o uso de pesticidas para que a sua produção agrícola não sofra grandes perdas por causa de pestes, larvas e vermes. A aplicação dessas substâncias é realizada por meio de um pulverizador, que possui canos laterais para o lançamento de líquido.

A quantidade de pesticida aplicada em cada região da terra cultivada, juntamente com a temperatura do ambiente pode influenciar a obtenção de sucesso na proteção contra as pestes. A perda de produção muitas vezes significa grandes perdas econômicas para o agricultor, assim, seria de extrema utilidade algum mecanismo a fim de controlar esse processo de maneira segura.

1.2.1 Solução Proposta

Considerando esse problema dos agricultores, nota-se que os mesmos necessitam de um dispositivo que meça o fluxo de pesticida e a temperatura do ambiente para uma posterior análise desses dados, caso necessário. Isso pode ser feito com dois sensores - um de fluxo de fluido e um de temperatura - conectados a um microcontrolador que envia esses dados a um servidor, associados à uma localização de GPS (*Global Positioning System*), a fim de relacioná-los à região em que o pesticida está sendo aplicado.

A transmissão desses dados deve ser feita de maneira segura, para que nenhum atacante externo consiga fazer alterações a fim de prejudicar a análise feita posteriormente

pelo agricultor. Isso pode ser feito com protocolos de aplicação combinados a protocolos de segurança para dispositivos IoT, como, respectivamente o CoAP e DTLS, os quais serão descritos nos capítulos 2 e 3.

1.3 Metodologia

O desenvolvimento deste trabalho segue 5 etapas: revisão bibliográfica, entendimento dos conceitos para aplicação, implementação, testes e análise dos resultados.

1.3.1 Revisão bibliográfica

Na revisão bibliográfica foram pesquisados trabalhos científicos, livros, artigos e especificações de protocolos relacionados com o tema deste trabalho, a fim de servir como base teórica para as etapas seguintes e contextualizar o leitor em relação à problematização.

1.3.2 Entendimento de aplicação dos conceitos

Tendo realizada a revisão bibliográfica, o próximo passo foi entender os conceitos pesquisados e escrevê-los na seção de fundamentação teórica, para assim conseguir apresentar uma solução do problema apresentado.

1.3.3 Implementação

Após encontrar um problema existente capaz de ser resolvido com os conceitos estudados, foi planejado e implementado um protótipo de solução a partir desses conceitos, envolvendo também dispositivos de IoT e softwares que auxiliaram a solução.

1.3.4 Testes

Com a implementação, a integração dos dispositivos de IoT e softwares foi testada para verificar que a solução pôde ser obtida com as tecnologias empregadas. O software teve que ser testado com diferentes entradas e os sensores, foram testados em diferentes condições, a fim de verificar se eles respondiam corretamente às mudanças das características do ambiente.

1.3.5 Análise

Os testes feitos na etapa anterior foram analisados e avaliados em termos de obtenção de sucesso na transmissão de dados, leitura de dados dos dispositivos e processamento do software.

1.4 Justificativa

Muitos dispositivos que usam IoT trazem grandes facilidades às nossas vidas e podem ser utilizados em contextos de grande importância, nos quais o sigilo é fundamental. Além disso, sensores ficam em lugares de fácil acesso e fácil alteração. Assim, o estudo e o desenvolvimento de técnicas relacionados com segurança são fundamentais, considerando as limitações dos dispositivos.

1.5 Objetivo geral

O objetivo geral deste trabalho é estudar protocolos de segurança específicos para dispositivos IoT e aplicar os conhecimentos obtidos em um estudo de caso.

1.6 Objetivos específicos

Os objetivos específicos deste trabalho são:

1. Pesquisar sobre protocolos e algoritmos de segurança em IoT.
2. Aplicar os protocolos estudados em um protótipo com comunicação segura.

1.7 Estrutura do documento

No capítulo 2, este trabalho aborda conceitos de IoT e alguns protocolos envolvidos na comunicação de dispositivos limitados. No capítulo 3, métodos de segurança, também relacionadas a IoT, serão abordados, com enfoque no DTLS (*Datagram Transport Layer Security*). Ainda no escopo de revisão bibliográfica, no capítulo 4 está a descrição de alguns trabalhos correlatos e um pequeno comparativo entre eles e este trabalho. O capítulo 5 descreve o desenvolvimento prático, contendo informações sobre o software e hardware usados, mas também os testes realizados. No capítulo 6 está a conclusão do trabalho.

2 Internet of Things

Como este trabalho trata de segurança no ambiente de IoT (*Internet of Things* ou Internet das Coisas), é fundamental esclarecer o que é e conceitos sobre essa rede de dispositivos. Inicialmente, a Internet era para uso de computadores pessoais e com a evolução dos microcontroladores a Internet das Coisas se tornou possível.

IoT é a estrutura que viabiliza a comunicação entre grandes servidores e computadores com pequenos e restritos microcontroladores. Engloba vários tipos de redes como: Redes de Sensores Sem Fio (WSN - *Wireless Sensor Network*), Máquina-a-Máquina (M2M - *Machine-to-Machine*), Redes Sem Fio de Baixo Consumo (LoWPAN - *Low-Power Wireless Personal Area Networks*) e Identificação com Rádio-Frequência (RFID - *Radio-Frequency Identification*).

Generalizando, IoT materializa a visão de rede na qual sensores terão a capacidade de se comunicar com outros dispositivos usando os protocolos da Internet (Granjal; Monteiro; Silva, 2015). Ou seja, IoT é composta por eletrônicos embarcados, softwares e sensores, que facilitam a integração entre o mundo físico e as redes de computadores, trazendo benefícios (Zhou et al., 2017).

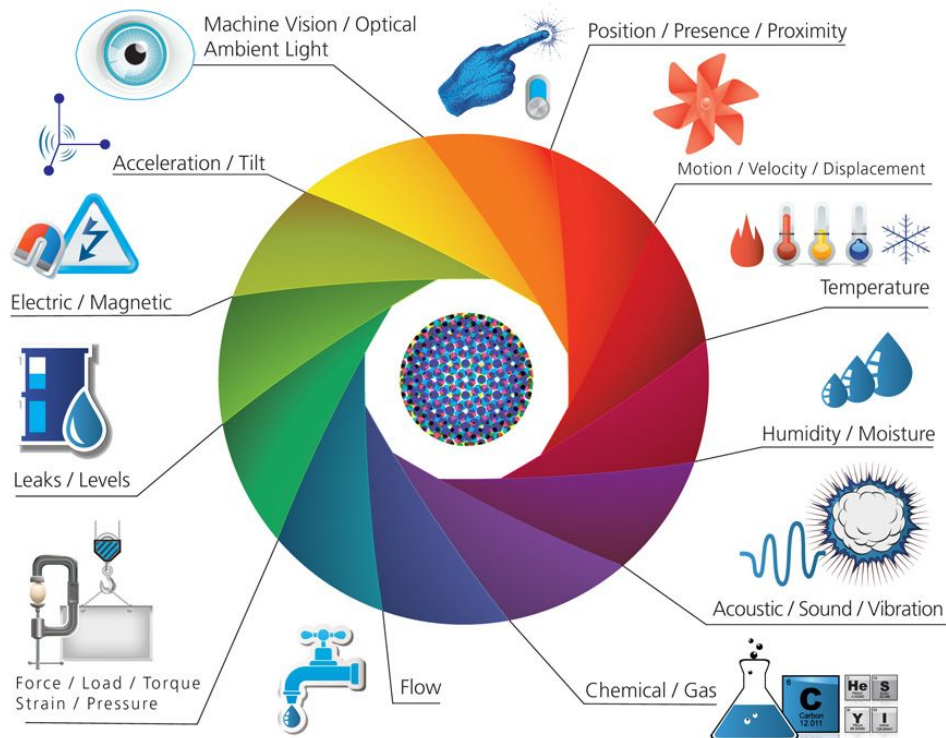
2.1 Aplicações

Diversas características do mundo físico são medidas através de sensores, pertencentes muitas vezes a aplicações de extrema importância. Pode-se observar alguns desses elementos na Figura 1. Sensores são capazes de medir intensidade luminosa, presença humana, velocidade, eletricidade, nível de líquidos, força e outros fatores, e por isso auxiliam aplicações.

Imagine um idoso vivendo sozinho em uma casa distante de seus familiares e, caso tenha algum problema repentino de saúde, não tenha a quem pedir ajuda rapidamente. Uma solução seria instalar sensores de movimento ou câmeras pelos cômodos de sua casa, que mediriam seus sinais vitais, e, ao detectar qualquer anormalidade, um sinal através da Internet seria enviado para uma ambulância a fim de obter assistência (Postcapes, 2015).

O trânsito lento de veículos, principalmente em horários de pico, é um problema que afeta diversas pessoas em diversas cidades do mundo, agravado pela falta de sincronização entre semáforos. Uma solução envolvendo IoT seria coletar informações sobre o fluxo de veículos através de câmeras e regular o período dos semáforos através da Internet. Além disso, o trânsito poderia sofrer alterações para facilitar o deslocamento de ambulâncias e viaturas policiais.

Figura 1 – Elementos medidos por sensores



Fonte: (Postcapes, 2015)

2.2 Limitações

É de extrema utilidade a capacidade de comunicação entre dispositivos, por mais diferentes que sejam, todavia adicionar um recurso a um dispositivo implica na geração de custos para essa adição. Mais energia é gasta com algoritmos - implementados em software ou hardware - para suporte de protocolos, que demandam espaço no chip ou em memória e no contexto de sensores esse custo pode ser alto, pois muitas vezes dispõe de apenas 10 KB de memória.

Esses fatores influenciam também na implementação de aspectos relacionados ao principal tema deste trabalho, a segurança voltada para dispositivos de IoT. Além dos recursos utilizados para o funcionamento do dispositivo, mais recursos como espaço em memória, energia e processamento são necessários para o funcionamento dos mecanismos de segurança.

2.3 Protocolos para IoT

Tendo em vista as limitações citadas, dispositivos limitados necessitam de protocolos específicos para conseguirem se comunicar com outras máquinas, utilizando a

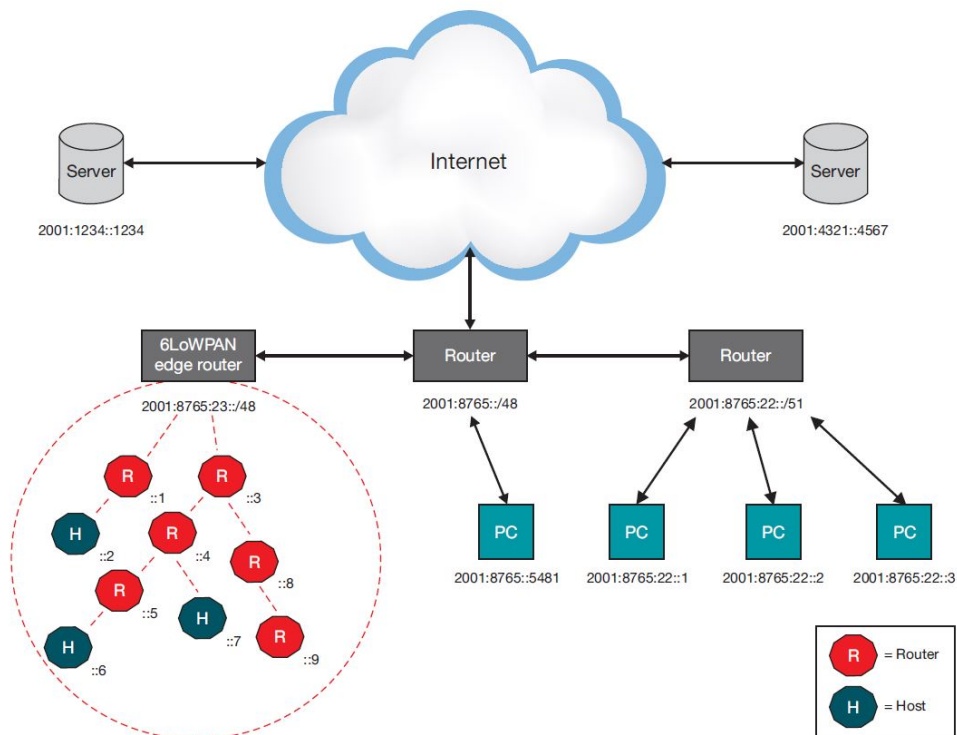
Internet, por exemplo. O fluxo de dados dos dispositivos pode ser sigiloso e exigir técnicas de segurança.

Considerando essa necessidade de suporte a comunicação, segurança e economia de espaço e energia, protocolos como 6LoWPAN (*Low-Power Wireless Personal Area Networks*), DTLS (*Datagram Transport Layer Security*) e CoAP (*Constrained Application Protocol*) foram criados.

2.4 6LoWPAN - Low-Power Wireless Personal Area Networks

Com o surgimento do IPv6 (Internet Protocol 6), o endereçamento de inúmeros dispositivos de IoT foi facilitado, porém não faz diferença existir a possibilidade de endereçamento de pequenos dispositivos se eles não possuem capacidade para execução do IPv6 (Olsson, 2014). Considerando isso, a IETF criou a 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*), a fim de permitir dispositivos com menores capacidades de processamento e pequenas memórias de se comunicarem via Internet.

Figura 2 – Topologia de uma rede com 6LoWPAN

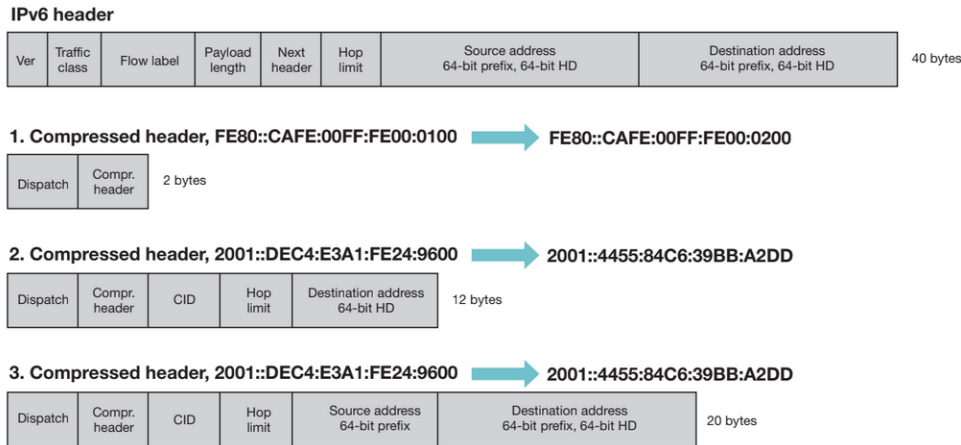


Fonte: (Olsson, 2014)

Como exibido na Figura 2, os dispositivos enviam pacotes de acordo com o protocolo e um roteador de saída específico (6LoWPAN *edge router*) trata a troca de informações entre os dispositivos e a Internet, mesmo que os destinos usem IPv4. Além disso, esse roteador cuida da comunicação dentro da própria rede de dispositivos.

O 6LoWPAN otimiza a transmissão de dados através das seguintes adaptações: compressão do cabeçalho, fragmentação, remontagem de dados e geração de endereço automática. Usando o 6LoWPAN, o cabeçalho de 40 bytes padrão no IPv6 é reduzido a no máximo 20 bytes, explorando redundâncias, usando compressão sem armazenamento de estado e permitindo que os protocolos de roteamento decidam a rota dinamicamente. Embora não mostrado na Figura 3, o cabeçalho referente ao UDP também sofre compressão.

Figura 3 – Compressão em relação ao IPv6

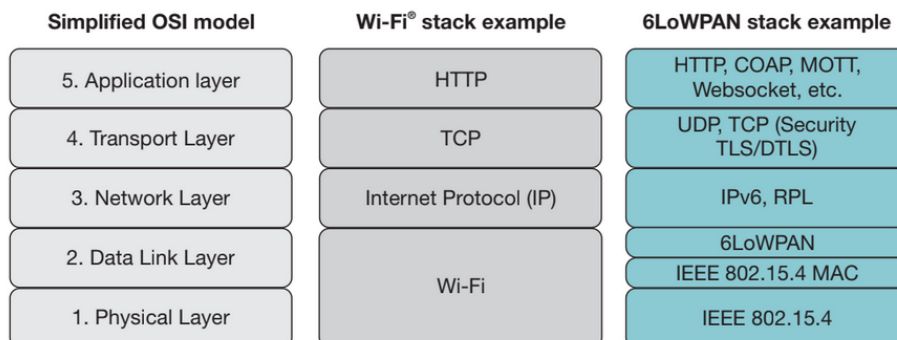


Fonte: (Olsson, 2014)

O primeiro caso de compressão representado na Figura 3 é válido apenas para troca de informações entre nodos vizinhos. Já no segundo caso, o prefixo da rede destino é conhecido e omitido do cabeçalho. No terceiro caso, o prefixo da rede destino não é conhecido e deve ser especificado, todavia o cabeçalho ainda assim é comprimido em 50%, em relação ao protocolo IPv6 padrão.

A fragmentação de quadros é outra adaptação para possibilitar o uso do IPv6 via links de rádio, no entanto informações para a ordenação precisam ser adicionadas e o processamento relacionado a fragmentação consome mais energia do dispositivo.

Figura 4 – Pilha de protocolos



Fonte: (Olsson, 2014)

A Figura 4 mostra a equivalência entre camadas em três modelos diferentes de rede: modelo OSI simplificado, pilha de protocolos Wi-Fi e a pilha de protocolos usando 6LoWPAN. Nos dois componentes superiores da pilha mais a direita estão protocolos como CoAP e DTLS, voltados para IoT e que serão explicados, respectivamente, nas seções 2.5 e 3.2.

Na base da pilha 6LoWPAN está o protocolo *IEEE 802.15.4*, específico para a camada física de dispositivos consumidores de pouca energia e, por consequência, com pequena taxa de transmissão, provendo flexibilidade e mantendo a compatibilidade com a Internet. A camada física mede também a qualidade do sinal e define qual dos 27 canais contidos no protocolo *IEEE 802.15.4* serão usados.

A pilha contém também o protocolo *IEEE 802.15.4 MAC*, referente a camada de enlace. Nele estão implementadas as funções de configurações para uma rede ponto-a-ponto entre dispositivos, de controle de acesso do canal evitando colisão (CSMA-CA), de entrega de pacote garantida e de uso intermitente do rádio, visando economia energética.

Na Figura 2 existem nodos folha (hosts) e nodos intermediários (routers). Os nodos folha podem operar em RFD (Reduced Function Device), pois não tem a função de encaminhar pacotes de outros dispositivos, economizando memória e energia. Os nodos intermediários operam em FFD (Full Function Device), porque, além de nodos, são coordenadores da rede.

2.5 CoAP - Constrained Application Protocol

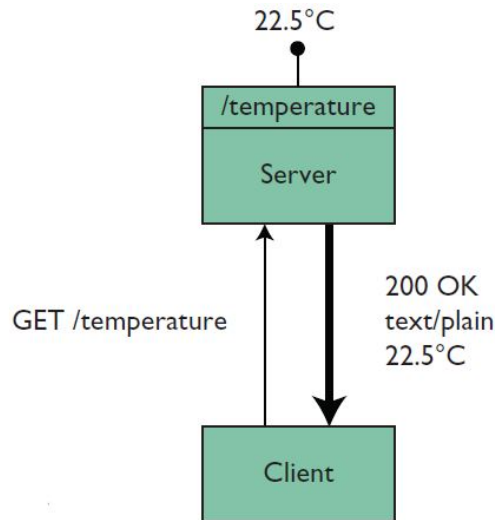
O HTTP (*HyperText Transfer Protocol*) passou por mais de uma década de crescimento organizado, acumulando cada vez mais implementações, o que supera a capacidade de ser usado em dispositivos pequenos (Bormann; Castellani; Shelby, 2012).

Protocolos de rede voltados a dispositivos pequenos, com 10 KB de RAM e 100 KB de código, são projetados para não haver sobrecarga. Na camada de aplicação, com essa configuração, não há espaço para bibliotecas HTTP no dispositivo, motivando a criação de protocolos como o CoAP. Microcontroladores com mais recursos conseguem executar o protocolo HTTP, no entanto podem ter seus recursos melhor aproveitados com o uso do CoAP.

A simplicidade do HTTP é atrativa através de comandos como GET, PUT, POST e DELETE, possibilitando ao cliente requisitar um parâmetro do sensor de maneira clara. Um exemplo está representado na Figura 5: a fim de obter a temperatura medida por um sensor em determinado momento, o cliente envia o comando GET /temperatura e recebe uma resposta informando que a requisição foi bem sucedida, o formato dos dados e os próprios dados. Além disso, a requisição é compatível com browsers web.

Apesar dessas vantagens, como já mencionado, o HTTP é custoso e usa como base outro protocolo extremamente custoso: o TCP. Embora ele esteja presente na pilha de protocolos relacionada ao 6LoWPAN na Figura 4, o TCP, apesar de confiável, possui grande “overhead” para dispositivos pequenos.

Figura 5 – GET HTTP

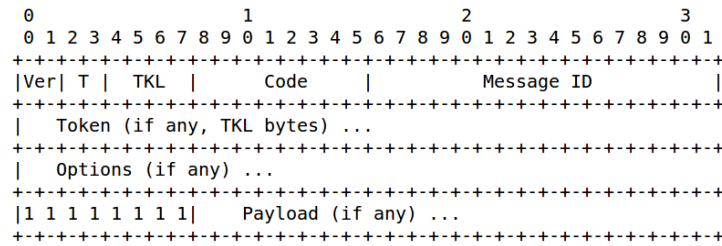


Fonte: (Bormann; Castellani; Shelby, 2012)

O CoAP é um protocolo de transferência de informações da camada de aplicação, com o mesmo conjunto de operações básicas se comparado ao HTTP, porém consideravelmente mais simples, consumidor de menos memória, processamento e, conseqüentemente, energia (Bormann; Castellani; Shelby, 2012). O principal fator responsável pela redução da complexidade é o uso do UDP ao invés do TCP. O CoAP trata o problema de perda de pacotes, não solucionado pelo UDP, adicionando uma camada à mensagem para detecção e retransmissão dos pacotes. Por esse motivo, um cabeçalho CoAP tem apenas quatro bytes somados a dois bytes para opções e a resposta da requisição ocupa apenas um byte. Assim, uma requisição geralmente usa de 10 a 20 bytes.

2.5.1 Cabeçalho

Figura 6 – Cabeçalho CoAP



Fonte: (Bormann, 2014)

A Figura 6 mostra o cabeçalho do CoAP e seus campos serão descritos a seguir.

2.5.1.1 Version

Indica a versão do protocolo usada, para que os dois lados da comunicação saibam qual versão do protocolo seguir. Caso um valor não conhecido esteja nesse campo, a mensagem deve ser ignorada.

2.5.1.2 Type

Confirmable (0): Mensagem enviada deve ser confirmada pelo receptor através de um “Acknowledgement”, para mensagens com confiabilidade. Essa mensagem é retransmitida com intervalos que crescem exponencialmente até o recebimento do “Ack” ou “Reset”.

Non-Confirmable(1): Mensagem enviada deve não ser confirmada com “Acknowledgement” e rejeitada caso o receptor não seja capaz de entendê-la.

Acknowledgement (2): Mensagem enviada como primeira resposta de uma mensagem do tipo Confirmable, identificando o ID da mensagem que está sendo confirmada, indicando seu recebimento.

Reset (3): Indica que uma mensagem (Confirmable ou Non-Confirmable) foi recebida, mas o receptor não pôde processar adequadamente, provavelmente por não conhecer o contexto adequado da mensagem.

2.5.1.3 Token Length

Indica o tamanho do campo Token, na segunda linha do cabeçalho.

2.5.1.4 Code

Indica o tipo de requisição que está sendo feita, semelhante ao HTTP, o CoAP implementa os métodos básicos GET, POST, PUT e DELETE. Tratando-se de uma

resposta, o campo Code indica o código da resposta pertencendo a uma das três classes: sucesso, erro do cliente ou erro do servidor.

2.5.1.5 Message ID

É um identificador, normalmente sequencial, para cada mensagem enviada por determinado emissor, o qual não deve ser reutilizado até o “Ack” não ser mais esperado pelo emissor da mensagem.

2.5.1.6 Token

Contendo de 0 a 8 bytes de informação útil, serve para relacionar a requisição à determinada resposta. Idealmente, deve ser implementado de maneira que o token para o par cliente-servidor seja único.

2.5.1.7 Options

É um campo usado para definir informações como: URI do destinatário da mensagem, porta do destinatário, caminho do recurso, query (em caso de parametrização), formato do payload, formato de conteúdo aceito como resposta, tempo limite para cache, condicionais de formato como resposta e outras.

2.5.1.8 Payload

O payload de uma mensagem é a representação do recurso requisitado ao servidor, em caso de GET, ou do recurso enviado ao servidor, em caso de POST ou PUT. Exemplificando: se o cliente fizer a requisição na URI "temperatura", e o servidor possuir a informação de que a temperatura é de 35°C, então essa informação será retornada no campo de payload.

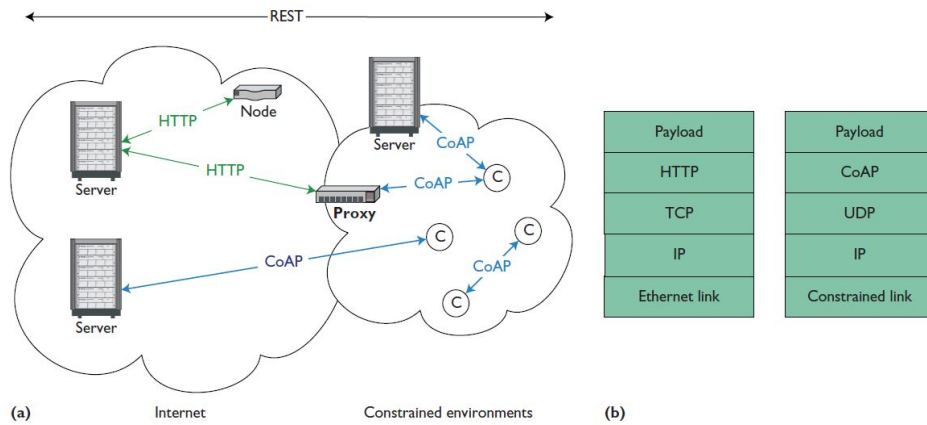
2.5.2 Integração com HTTP

A implementação do CoAP não se atém a um protocolo eficiente para a camada de aplicação, mas supera essa ideia viabilizando uma fácil compatibilidade com requisições HTTP através de *proxies*. Um *proxy* é um intermediário com um gateway, que com um lado se comunica usando certo protocolo e com outro lado se comunica usando um protocolo diferente, como pode-se observar na Figura 7 (Bormann; Castellani; Shelby, 2012).

Dessa maneira, quem faz a requisição consegue recuperar as informações desejadas transparentemente, sem ter conhecimento sobre o uso do CoAP pelo sensor. Isso permite a adesão ao CoAP pelo microcontrolador, permitindo o uso do HTTP por parte de outros dispositivos que desejarem enviar requisições. Apesar da compatibilidade entre HTTP e

CoAP, os dois dispositivos envolvidos na comunicação podem utilizar exclusivamente o CoAP.

Figura 7 – Compatibilidade entre CoAP e HTTP via proxy



Fonte: (Bormann; Castellani; Shelby, 2012)

Essa compatibilidade é devido à implementação do CoAP seguir a arquitetura REST (*Representational State Transfer*), na qual recursos são representados por URIs e são manipulados através de comandos como GET, POST, PUT e DELETE. REST foi descrita por um dos criadores do HTTP e estabelecida como um modelo para facilitar a integração de aplicações através da Web (Fielding, 2000).

2.5.3 Transmissão de dados grandes

Leituras em sensores de temperatura, por exemplo, não costumam exigir muitos bytes de dados, sendo suficiente o tamanho provido pelo pacote UDP. Todavia, é vantajosa para a manutenção dos sensores a disponibilidade de operações além de simples leituras providas pelo CoAP. O UDP não garante ordenação dos pacotes enviados, então transmissões muito grandes formadas por mais de um pacote seriam desordenadas em situações como, por exemplo, uma atualização de firmware.

Para resolver esse problema o CoAP possui uma versão com transmissão em blocos. Cada bloco é enviado com um identificador e uma flag informando se ele é o último bloco. Cada envio é respondido por um ACK, para não permitir perdas (Bormann, 2016).

2.5.4 Função Observador

Na arquitetura REST, normalmente, sempre quando um cliente deseja receber uma informação do servidor essa ação deve ser feita através de um comando GET para cada leitura. Dessa maneira, caso o cliente deseje receber uma informação periodicamente, ele deve enviar um GET periodicamente. No entanto, dispositivos com restrições energéticas

permanecem hibernando por muito tempo e o momento em que o comando GET é enviado pode coincidir com o momento de hibernação. Além disso, lidar com GETs periodicamente sobrecarrega o dispositivo.

Para não haver perdas de requisições, nem sobrecarga de dispositivos, o CoAP tem a Função Observador. Com esta função, o cliente envia um GET, com uma flag indicando o desejo de atualização periódica, ao servidor (sensor, microcontrolador e etc). O sensor, então, dorme sem perder requisições e envia periodicamente o dado especificado no GET.

2.5.5 Segurança

Assim como o HTTP é combinado com o TLS (*Transport Layer Security*) com objetivo de obter segurança nas trocas de mensagens, o CoAP é combinado com o DTLS (*Datagram Transport Layer Security*), para obter segurança semelhante à presente na Web e seu funcionamento será explicado na seção 3.2.

3 Segurança em IoT

A função de sensores e microcontroladores inseridos em uma rede IoT é enviar informações a um computador interessado em suas capturas. Todavia existe a possibilidade das informações desse tráfego serem extremamente sigilosas, gerando uma necessidade por métodos de segurança nas redes de sensores.

Assim como existem restrições de memória e processamento para os protocolos de comunicação, as ferramentas de segurança também sofrem com as mesmas limitações. Algoritmos complexos, que ocupam muita memória, ou arquivos relacionados à segurança, como certificados armazenadores de chaves, não são suportados por muitos dispositivos de IoT, que necessitam de abordagens eficientes para obterem segurança em suas atividades.

3.1 Segurança a partir de gateway

Uma solução seria implementar criptografia a partir de um gateway de saída da rede. Assim, a transmissão de dados entre sensores ocorreria sem criptografia e um gateway usaria técnicas clássicas de segurança - RSA (*Rivest-Shamir-Adleman*), AES (*Advanced Encryption Standard*), *Diffie-Hellman* e certificados - para transmissão segura.

Embora essa solução evitasse sobrecarga em sensores sem muita capacidade de processamento e memória abundante, vulnerabilidades ainda estariam presentes, como por exemplo a troca de um sensor ou leituras diretamente a partir dos sensores, caso o atacante esteja próximo dos sensores.

3.2 DTLS - Datagram Transport Layer Security

Um dos protocolos de segurança computacional mais usados é o TLS (Transport Layer Security), o qual possibilita uma série de configurações como algoritmo de troca de chaves, algoritmo de criptografia assimétrica, algoritmo de criptografia simétrica, modo de operação e algoritmo de hash (Dierks, 2008). Contém também envios de certificados X.509 relacionados a criptografia assimétrica e além disso, o TLS troca informações pela rede usando o TCP, de forma a obter garantia de recebimento. Como já citado, o TCP é evitado em dispositivos pequenos com pouca capacidade energética e de processamento, devido ao seu custo e por isso nesse contexto há preferência pelo UDP, causando uma incompatibilidade com o TLS.

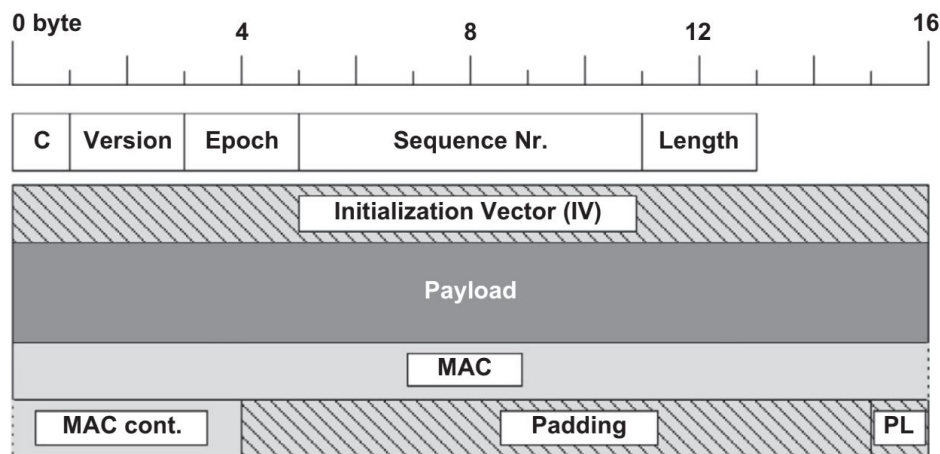
Em muitos casos, a maneira mais desejável para garantir segurança entre cliente e servidor seria usando o TLS, todavia o uso do UDP impossibilita o uso do TLS, por isso

o DTLS (TLS para datagramas) foi projetado para ser o mais similar ao TLS possível, minimizando novas invenções de seguranças e possibilitando aproveitamento de código e estrutura do TLS (Rescorla, 2012).

3.2.1 Registro do DTLS

Como visto na Figura 8, o DTLS possui um cabeçalho identificando o conteúdo C da mensagem, versão do DTLS usado, a época da mensagem, um número de sequência explícito e o tamanho da mensagem.

Figura 8 – Registro DTLS



Fonte: (Kothmayr et al., 2013)

3.2.1.1 Tipo de Conteúdo da Mensagem

Como no TLS, uma mensagem pode representar quatro situações diferentes: estabelecimento do *handshake*, mudança de protocolos de segurança, alerta de possível comprometimento de segurança ou uma mensagem da aplicação.

3.2.1.2 Versão do DTLS

Versões diferentes do DTLS possuem diferentes recursos, algoritmos e maneiras de tratar erros. Por isso, no cabeçalho existe o campo para indicar a versão do protocolo entre duas entidades, a fim de haver compatibilidade no uso do DTLS.

3.2.1.3 Número de sequência

A principal função do número de sequência é a reordenação de mensagens. No DTLS, cada mensagem recebe um número de sequência específico e quando um dispositivo recebe uma mensagem X ele consegue determinar se X é a próxima mensagem esperada.

Se for, ela é processada. Se não for, X é direcionada a uma fila para ser processada no momento correto (Rescorla, 2012).

3.2.1.4 Época da mensagem

A época da mensagem é um campo incrementado a cada troca de configurações de segurança (*cipher state change*) e serve para mensagens serem interpretadas de acordo com as configurações adequadas. Durante a transmissão, os pacotes sofrem alteração de ordem, sendo posteriormente reordenados pelo número de sequência.

O *handshake* geralmente ocorre na época 0 e ao final dele, começa a época 1 com configurações de segurança, algoritmos de cifra simétrica e hash, estabelecidas. A última X mensagem do *handshake*, com época 0, pode ser perdida, exigir retransmissão e ser entregue após a primeira mensagem Y da época 1. Embora o número de sequência da mensagem X seja maior comparado ao número de sequência de Y, a especificação de época permite que a mensagem X seja considerada mais antiga em relação a Y e também permite a leitura de X com os parâmetros de segurança da época 0.

3.2.1.5 Vetor de Inicialização

Cifras de bloco possuem modos de operação como o CBC (*Cipher Block Chaining*), no qual a saída de um bloco é usado como entrada para cifrar o bloco seguinte. Na cifra do primeiro bloco é usado um vetor de inicialização, geralmente aleatório, necessário para o destinatário decifrar a mensagem.

3.2.1.6 Payload

O payload é a mensagem em si, que será interpretada pela aplicação após decifração.

3.2.1.7 MAC

Assim como no TLS, o MAC é calculado para verificação de integridade e autenticidade da mensagem enviada. Todavia, diferentemente do TLS, no DTLS um erro de MAC implica em um alerta e um descarte da mensagem, sem término de conexão. O MAC é calculado com base em uma chave e a concatenação dos campos presentes na Figura 8, considerando o payload em texto claro e excluindo os campos MAC, MAC cont, padding e PL.

3.2.2 Handshake

O *handshake* consiste em uma sequência de troca de mensagens, de forma que, ao fim dessa troca, as configurações de segurança entre cliente e servidor estão estabelecidas.

3.2.3 Cipher Suite

Tanto no TLS quanto do DTLS o Cipher Suite define o conjunto de algoritmos criptográficos que serão utilizados para determinadas ações ao longo da sessão entre os dois lados envolvidos na comunicação segura. O Cipher Suite é representado por uma string como por exemplo: **TLS_DHE_RSA_WITH_AES_128_CBC_SHA256**.

3.2.3.1 Algoritmo de troca de chaves

O algoritmo de troca de chaves é o primeiro que aparece e no caso da string exemplo o algoritmo escolhido foi o **DHE** (Diffie-Hellman efêmero). A chave trocada será usada como base para os algoritmos de cifra simétrica e HMAC.

3.2.3.2 Algoritmo de autenticação

É o algoritmo para a garantir que o servidor com quem o cliente se comunica é realmente quem ele diz ser. Isso ocorre com o auxílio de um certificado pertencente exclusivamente ao servidor assinado por uma autoridade certificadora. Dessa maneira, a autoridade certificadora garante, através do do certificado do servidor, que o a chave pública guardada no certificado é autêntica. Considerando o exemplo, o algoritmo de autenticação é o **RSA**, para esse Cipher Suite.

3.2.3.3 Algoritmo de cifra simétrica

O algoritmo de cifra simétrica é o utilizado para cifrar as mensagens trocadas entre cliente e servidor. Nele, a chave, gerada a partir da troca de chaves, é a mesma tanto no servidor quanto no cliente, por isso é chamado de simétrico. Exemplos de algoritmos de cifra simétrica são: AES, DES, 3DES, Blowfish e RC4. No exemplo, é usado o **AES**, com blocos de **128** bits e em modo **CBC** (Cipher Block Chaining).

3.2.3.4 Algoritmo de HMAC

O algoritmo de HMAC utiliza um algoritmo de resumo criptográfico - como **SHA256** - utilizado no Cipher Suite exemplo, para gerar um resumo da mensagem enviada. Dessa forma, o receptor da mensagem pode verificar se houve alguma alteração, realizada por terceiros, na mensagem recebida, gerando um resumo criptográfico e comparando com o recebido.

4 Trabalhos correlatos

4.1 Trabalho 1: Constrained Application Protocol (CoAP) no Arduino UNO R3: Uma Análise Prática

O primeiro trabalho correlato (Porciúncula et al., 2018) trata do uso do CoAP como protocolo de comunicação em dispositivos restritos, ou seja, com pouca capacidade de processamento, energia e memória. Esses dispositivos são classificados em três categorias, de acordo com sua capacidade de armazenamento de dados e de código.

O experimento dos autores envolve um Arduino UNO conectado à Internet por um cabo Ethernet, usando bibliotecas disponíveis para Ethernet em Arduino, se comunicar com um servidor CoAP instalado em um notebook na mesma rede. O Arduino utilizou a biblioteca `microcoap` e o servidor usou a biblioteca `CoAPthon`. Os autores comprovaram a comunicação usando o `WireShark` para capturar pacotes do protocolo.

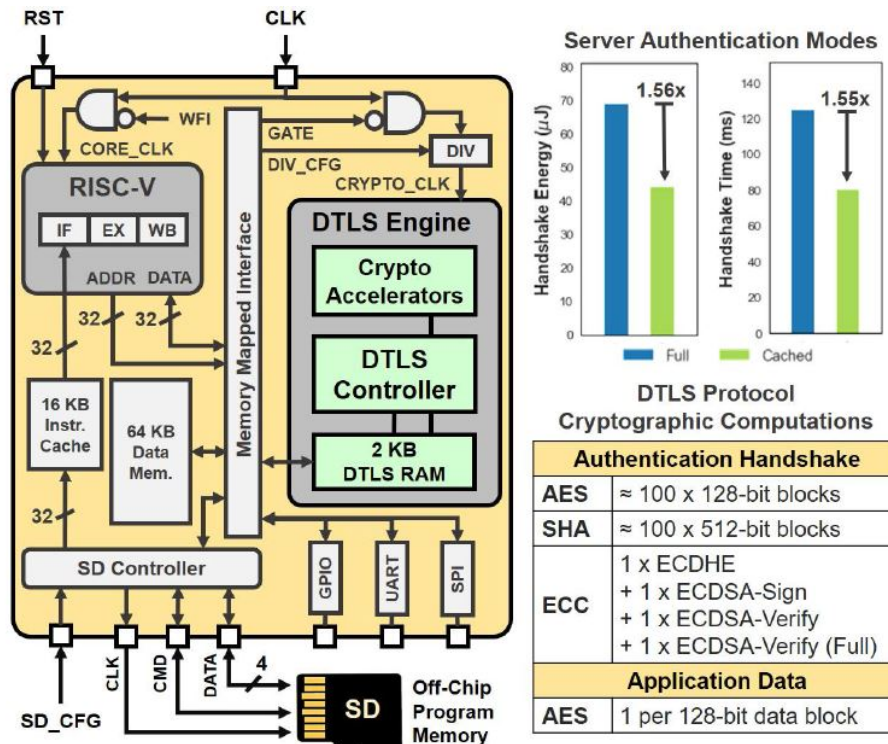
O trabalho correlato mencionado, porém, embora cite, não lida com os protocolos de segurança em seus experimentos, diferentemente deste trabalho, que cita e explora a integração do CoAP com protocolos de segurança de maneira prática.

4.2 Trabalho 2: An energy-efficient reconfigurable DTLS cryptographic engine for End-to-End security in iot applications

Os autores do trabalho “An energy-efficient reconfigurable DTLS cryptographic engine for End-to-End security in iot applications” (Banerjee et al., 2018) abordam o problema de segurança em dispositivos restritos com um hardware específico que implementa alguns modos do protocolo DTLS. A justificativa, segundo eles, é que o DTLS exclusivamente implementado em software consome grande espaço em memória e é mais lento, caso comparado a uma implementação em hardware. Só o processamento de criptografia de curvas elípticas (ECC), ocupa 99% do tempo da fase de *handshake* do DTLS, assim, é vantajoso um acelerador criptográfico para melhorar o desempenho do protocolo.

O chip desenvolvido, mostrado na Figura 10 possui basicamente um processador RISC-V para implementação de outros protocolos, um mecanismo de processamento para o DTLS e portas de comunicação por protocolos como UART (*Universal asynchronous receiver/transmitter*) e SPI (*Serial Peripheral Interface*).

Figura 10 – Chip proposto

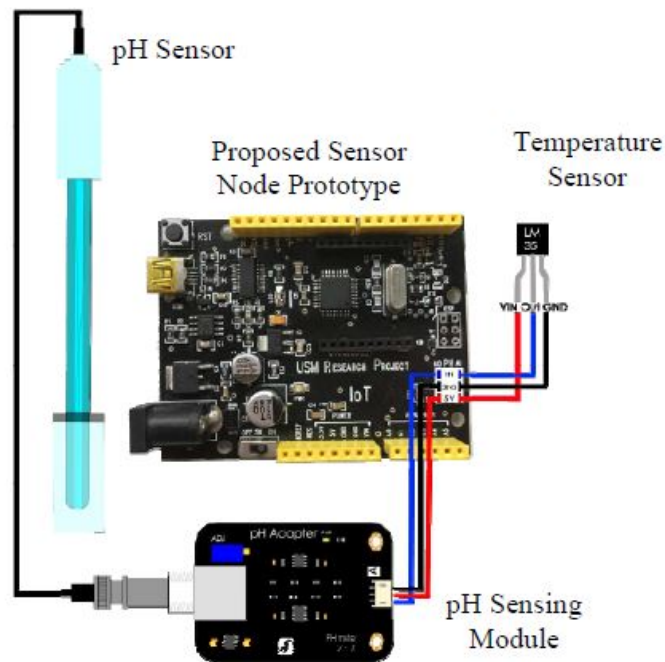


Fonte: (Banerjee et al., 2018)

4.3 Trabalho 3: Water quality monitoring with internet of things (IoT)

Esse trabalho correlato (Kamaludin; Ismail, 2017) propõe uma maneira de medir a qualidade da água e informar as características desejadas a um servidor, por meio de sensores e um microcontrolador, exibidos na Figura 11. A implementação foi feita com um microcontrolador Atmega328p, compatível com Arduino, ligado a um sensor de pH, um sensor de temperatura e uma antena WSN (Wireless Sensor Networks).

Figura 11 – Sensor de pH e temperatura conectados ao controlador



Fonte: (Kamaludin; Ismail, 2017)

Após a captura o microcontrolador envia os dados por uma antena até uma base de dados, onde um Arduino também ligado a uma antena recebe as informações e envia à Internet através do Arduino Ethernet Shield. À medida que os dados sobre a água são enviados, eles podem ser lidos a partir de um celular com sistema operacional Android por meio de um aplicativo desenvolvido pelos autores.

Em termos de resultado, o pH pôde ser medido com sucesso em um lago da universidade e foram realizadas também análises de consumo energético e alcance da antena para WSN.

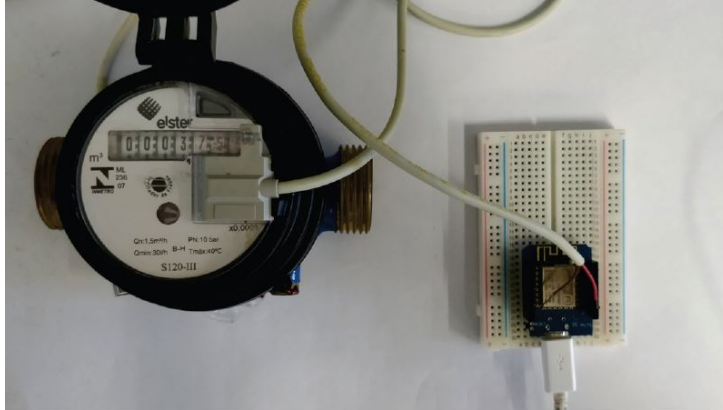
O trabalho correlato 3 possui semelhanças com o presente trabalho, todavia ele não transmite as informações capturadas de maneira segura. Enquanto os autores deste trabalho correlato medem pH e temperatura da água, este trabalho mede o fluxo de fluido no pulverizador de pesticidas e a temperatura do ambiente.

4.4 Trabalho 4: Telemetry for domestic water consumption based on IoT and open standards

O quarto trabalho correlato (Tavares et al., 2018) visa diminuir o desperdício de água com auxílio de dispositivos de IoT e sensores. Para isso, usou um sensor de fluxo de fluídos conectado a um módulo WiFi (ESP8266), semelhante aos hardwares usados neste

presente trabalho, e assim, o fluxo de água em um cano é enviado via protocolo HTTP a um servidor. O hardware descrito pode ser visto na Figura 12.

Figura 12 – Sensor de fluxo e ESP8266 conectados



Fonte: (Tavares et al., 2018)

Com base nesses dados, o servidor destino consegue detectar vazamentos, informar o consumo de água em determinada hora e, se essa tecnologia for empregada em uma empresa de fornecimento de água, um valor de cobrança pode ser calculado.

4.5 Comparação com o trabalho presente

O presente trabalho agrupa conceitos abordados nos quatro trabalhos correlatos descritos, envolvendo monitoramento de determinadas características físicas do ambiente, como apresentado nos trabalhos correlatos 3 e 4. Embora não usando hardware criptográfico, como desenvolvido no trabalho correlato 2, o presente trabalho utiliza o protocolo DTLS por meio de software para garantir uma comunicação segura por meio do CoAP, o qual também foi utilizado no trabalho correlato 1, por meio do Arduino UNO R3.

Trabalho	Descrição	Leitura por sensores	Utiliza CoAP	Utiliza DTLS
Presente trabalho	Envio seguro de dados relacionados a aplicação de pesticidas através dos protocolos CoAP e DTLS	Sim	Sim	Sim
(Porciúncula et al., 2018)	Transmissão de dados utilizando CoAP no Arduino UNO	Não	Sim	Não
(Banerjee et al., 2018)	Projeto de um hardware criptográfico para implementar o DTLS	Não	Não	Sim
(Kamaludin; Ismail, 2017)	Monitoramento da qualidade da água através de sensores	Sim	Não	Não
(Tavares et al., 2018)	Monitoramento do consumo de água através de sensores	Sim	Não	Não

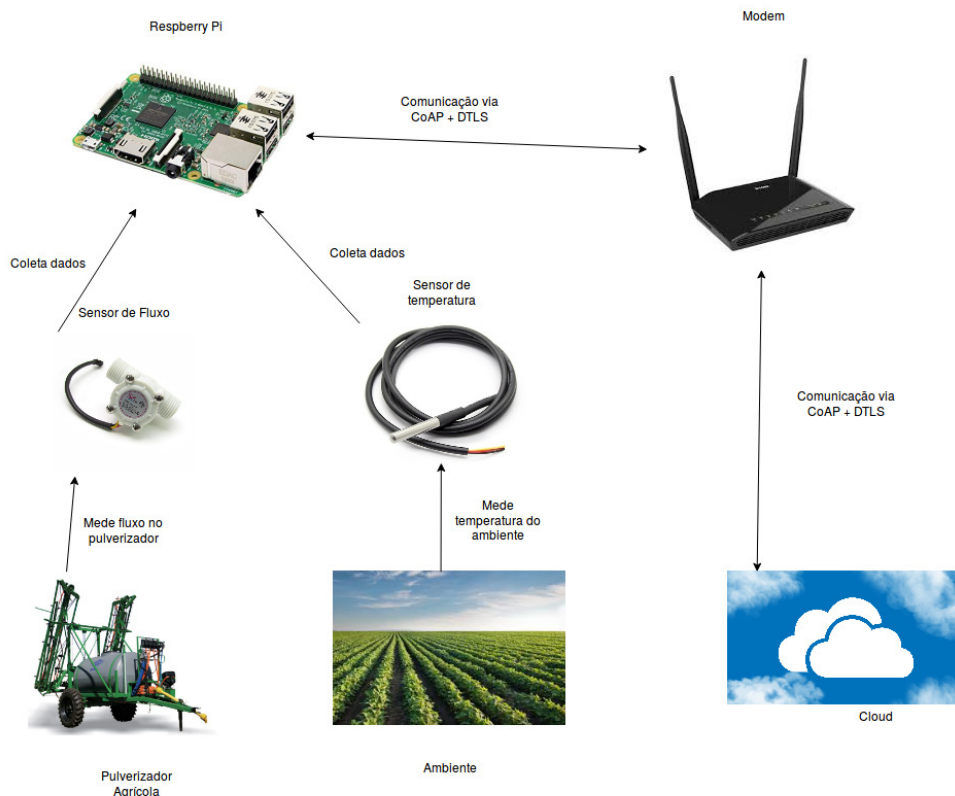
Tabela 1 – Comparação entre trabalhos correlatos e trabalho presente

5 Desenvolvimento de uma Aplicação com dispositivos IoT usando DTLS e CoAP

5.1 Ambiente usado como prova de conceito

A prova de conceito foi baseada no ambiente representado na Figura 13 que envolve a comunicação de dados entre dispositivos de pequeno porte. O sensor de fluxo faz leituras do fluxo de pesticida no pulverizador agrícola ao longo do tempo e o sensor de temperatura faz leituras da temperatura de determinado local do terreno em que o pesticida está sendo aplicado. Para associar o fluxo e a temperatura em determinado local do terreno deve ser usado um dispositivo de GPS, que não está representando na figura pois não foi usado neste desenvolvimento. O sensor de fluxo também não foi testado no desenvolvimento deste trabalho.

Figura 13 – Uso de IoT em aplicação agrícola



Fonte: Elaborada pelo autor

Após a coleta dos dados, a Raspberry Pi envia essas informações para a Cloud através de um gateway (modem representado na Figura 13) de maneira segura, utilizando

o CoAP como protocolo da camada de aplicação e o DTLS como protocolo da camada de segurança. A Raspberry Pi se comporta como um cliente CoAP, enquanto a Cloud se comporta como servidor, pois basta a Raspberry Pi conhecer o endereço do servidor e enviar dados. A Cloud, como servidor, é responsável por processar e armazenar os dados obtidos.

Quanto aos protocolos de comunicação entre dispositivos, os sensores de fluxo e temperatura se comunicam com a Raspberry Pi por meio de um único fio cada e a Raspberry Pi se comunica com o gateway via WiFi.

A prova de conceito, representada pelo ambiente e implementações previstas na Figura 13, foi desenvolvida e testada para verificação do funcionamento do DTLS e suas características de segurança, no contexto de Internet das Coisas.

5.2 Hardware usado

A fim de obter os dados necessários para o agricultor, três dispositivos são necessários: um sensor de temperatura, um sensor de fluxo e um dispositivo para captar as informações dos sensores e enviar via Internet. Cada dispositivo é descrito na sequência do texto.

5.2.1 Sensor de temperatura

O sensor de temperatura que serve para capturar a temperatura do ambiente é o DS18B20 (Figura 14), com um grande suporte para arduino em termos de implementações e informações. Esse sensor é a prova da água, mede temperaturas no intervalo de -55°C até 125°C , usa apenas um pino digital para comunicação e possui precisão de 0.5°C entre -10°C e 85°C .

A comunicação com apenas um pino através do protocolo 1-Wire é vantajosa pois permite que vários dispositivos escravos usem o mesmo pino de maneira organizada, economizando conexões.

Figura 14 – DS18B20



Fonte: (Thomsen, 2015)

5.2.2 Sensor de fluxo

Para medir o fluxo do pulverizador de pesticida, foi usado o sensor de fluxo YF-S201b (Figura 15), o qual possui uma precisão de 10% medindo em uma faixa de 1 L/min até 30 L/min, operando entre -25°C e 80°C . Em termos de comunicação, o sensor manda um sinal PWM (Pulse Width Modulation), que tem a frequência do sinal aumentada à medida que o fluxo aumenta. PWM é vantajoso pois é econômico, salva espaço de armazenamento e é imune a ruídos (Barr, 2001).

Mecanicamente, o sensor funciona com pás que giram com o fluxo de fluídos e quanto maior o fluxo, ocorrem mais rotações por segundo.

Figura 15 – Sensor de fluxo



Fonte: (Chris, 2015)

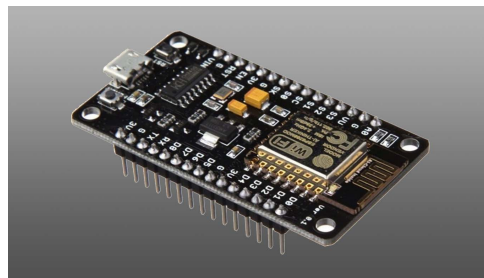
5.2.3 NodeMCU

Embora não sendo utilizado na versão final do desenvolvimento, o NodeMCU foi utilizado nos testes iniciais envolvendo o protocolo CoAP, descritos na seção 5.3.1, justificando a apresentação desse dispositivo.

O NodeMCU, apresentado na Figura 16, é uma placa da família do ESP8266, com código aberto, no repositório "<https://github.com/esp8266/Arduino>", e capaz de se conectar com a Internet via WiFi. Além da possibilidade de conexão (via protocolos UDP e TCP), seu repositório implementa funcionalidades de cliente HTTP, servidor HTTP, servidor de DNS, modo access point de WiFi e outras.

O NodeMCU suporta diferentes ambientes de programação como Arduino, MicroPython, LuaNode, NodeMCU, Esp7266 Basic e Espruino (JavaScript). Nos testes realizados foi usada a Arduino IDE para codificar e enviar o código compilado ao NodeMCU.

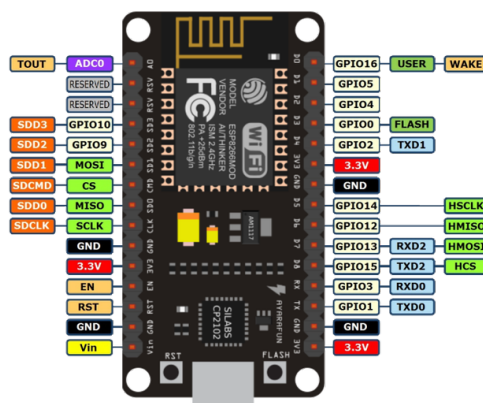
Figura 16 – Nodemcu



Fonte: (Malan, 2017)

Além das funcionalidades relacionadas à Internet e rede wireless, o Nodemcu possui um conjunto de pinos, como mostra a Figura 17. Existem pinos digitais, pinos de ground, pinos de alimentação (3v3), um pino analógico, um pino de enable e um de reset. Pode-se observar dois botões na parte inferior da imagem, um tem a função de reiniciar o programa presente na memória (RST) e outro auxilia o envio de código à memória flash do dispositivo.

Figura 17 – Pinos físicos



Fonte: (Singh, 2016)

5.2.4 Raspberry Pi 3

Raspberry Pi foi desenvolvido no Reino Unido com o objetivo de funcionar como um computador de baixo custo, para incentivar as pessoas a interagir com computadores e facilitar a aprendizagem de programação. Possui portas comuns aos computadores tradicionais, como USB, micro USB, HDMI, P2 e Ethernet (Wallace; Richardson, 2016).

Figura 18 – Raspberry Pi



Fonte: (Desconhecido, 2016)

A Figura 18 mostra o hardware da Raspberry Pi com as portas anteriormente citadas além de outros componentes (Gonçalves, 2018). Os principais componentes da Raspberry Pi são os seguintes:

- Processador: 1.2GHz QUAD Core Broadcom BCM2837 64bit ARMv8
- 1GiB de RAM
- 4 Portas USB
- 1 porta de vídeo HDMI

- 1 porta de vídeo RCA
- 1 porta ethernet
- 1 porta MicroSD
- WiFi e Bluetooth BCM43438
- 40 pinos GPIO

Os pinos GPIO permitem a conexão da Raspberry Pi com outros componentes como os sensores de temperatura e de fluxo utilizados neste desenvolvimento, tanto para alimentação energética quanto para a transferência de dados.

Em termos de sistema operacional, a Raspberry Pi já foi utilizada em projetos com uma grande variedade de sistemas operacionais como Debian, Windows 10 IOT e Plan 9, o que facilita o trabalho do desenvolvedor, pois programas desenvolvidos para computadores pessoais podem ser reaproveitados em projetos envolvendo a Raspberry Pi. No desenvolvimento desse trabalho, por exemplo, inicialmente seria usado o NodeMCU, todavia por dificuldades em adaptar a biblioteca com DTLS para essa plataforma, foi decidido usar a Raspberry Pi.

Por causa dessas facilidades e por funcionar como um computador, tendo um tamanho pequeno e ainda com pinos GPIO para se comunicar com sensores, a Raspberry Pi é considerada concorrente às placas da família Arduino, as quais sempre foram muito populares em projetos com sensores e IoT.

5.3 Bibliotecas

Para a execução da parte prática deste trabalho, foram utilizadas e adaptadas bibliotecas de software existentes.

5.3.1 CoAP: biblioteca ESP-CoAP

Uma implementação do CoAP específica para os módulos WiFi ESP8266 desenvolvida pelos autores (Grokhotokov; Molinari, 2018) foi encontrada no github. A biblioteca implementa tanto a parte do cliente, quanto a parte do servidor.

Inicialmente foi feita a comparação entre a especificação do protocolo descrita no documento RFC 7252 (Bormann, 2014) e a implementação do CoAP obtida em (Grokhotokov; Molinari, 2018).

Após a verificação de que a implementação estava de acordo com a especificação RFC 7252, foi realizado um teste a partir do dispositivo Nodemcu, realizando uma ope-

ração de GET no servidor de testes "coap://coap.me:5683" na URI "/test" e uma resposta foi obtida, indicando o funcionamento da implementação.

Inicialmente o desenvolvimento seria realizado com a placa Nodemcu esp8266, então a primeira implementação do CoAP utilizada foi específica para esse modo e os testes iniciais foram feitos com esta placa. Com a mudança de hardware para um dispositivo da família Raspberry Pi, uma nova implementação foi utilizada, que será descrita na seção seguinte.

5.3.2 CoAP e DTLS: biblioteca CoAPthon

CoAPthon é uma biblioteca em python para o protocolo CoAP (Tanganelli, 2018; Tanganelli; Vallati; Mingozzi, 2015).

Como o próprio autor da biblioteca informa (Tanganelli; Vallati; Mingozzi, 2015), CoAPthon é uma implementação de CoAP usando a linguagem Python que possui as funcionalidades descritas no documento RFC 7252 (Bormann, 2014) do CoAP. Assim como a biblioteca ESP-CoAP, CoAPthon implementa as operações básicas do CoAP, tanto no papel de cliente quanto no papel de servidor, todavia supera a ESP-CoAP ao implementar a função de, por exemplo, utilizar proxy entre CoAP e HTTP, como descrito na seção 2.5.2.

Outra vantagem da CoAPthon é a presença de uma implementação do DTLS para utilização conjunta ao CoAP caso desejado pelo usuário da biblioteca. Durante o processo de pesquisa de implementações envolvendo CoAP com DTLS, foi difícil encontrar uma biblioteca que de fato permitia o CoAP operar com o DTLS.

5.4 Implementação e testes

O autor da biblioteca CoAPthon disponibiliza no seu repositório do GitHub (Tanganelli, 2018) alguns testes envolvendo CoAP em conjunto com o DTLS no arquivo "test_secure.py". Por se tratar de um teste, tanto cliente e servidor foram instanciados no mesmo arquivo de código. Tendo em vista esse fato, foram feitas adaptações e modificações no código existente, sendo que as principais tarefas do desenvolvimento foram:

- Criar código da parte cliente e da parte servidor envolvendo a união dos protocolos CoAP e DTLS, com base nos testes realizados pelo autor da biblioteca.
- Desenvolver códigos para leitura de informações dos sensores de temperatura.
- Desenvolver código para unir a leitura de sensores e enviar os dados utilizando CoAP com DTLS.

- Desenvolver código para o servidor salvar as informações recebidas em um banco de dados.
- Desenvolver código para lidar com as requisições enviadas ao servidor (POST, GET, DELETE e PUT).
- Desenvolver código para unir operações do banco de dados, criação do servidor CoAP com DTLS e o tratamento de requisições.
- Implantar os códigos relacionados ao servidor em uma Cloud da Google.
- Implantar os códigos relacionados ao cliente na Raspberry Pi 3.

5.4.1 Desenvolvimento da parte servidor

Tendo como base o método “setUp” do arquivo “test_secure.py”, todos os métodos invocados pelo “setUp” (direta ou indiretamente), chaves e certificados usados e todas as inclusões de outras bibliotecas, que lidam com a criação de um servidor CoAP, foram copiados e adaptados em um arquivo próprio chamado “coapdtlsserver.py”.

A classe que representa o servidor é chamada de CoAPDtlsServer, que recebe como parâmetro o IP e a porta utilizada pelo servidor (Listagem A.1), para assim começar a configuração dos certificados e do socket utilizado no DTLS. O socket então é passado como parâmetro para a instanciação do servidor CoAP. É importante ressaltar que para propósitos de teste o socket é configurado com um certificado assinado por uma autoridade certificadora fictícia.

Listagem 5.1 – Inicialização do servidor

```

1  def __init__(self, host, port):
2      self.host_address = (host, port)
3
4      self.pem = self._setUpPems()
5
6      # Set up a server side DTLS socket
7      _sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8      _sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
9      _sock = wrap_server(_sock,
10                         keyfile=self.pem['SERVER_KEY'],
11                         certfile=self.pem['SERVER_KEY'],
12                         ca_certs=self.pem['CA_CERT'])
13     _sock.bind(self.host_address)
14     _sock.listen(0)
15
16     # Connect the CoAP server to the newly created socket

```

```

17     self.server = CoAPServer(self.host_address,
18                             starting_mid=None,
19                             sock=_sock,
20                             cb_ignore_listen_exception=self._cb_ignore_listen_exception)
21     self.server.add_resource('gpsflowtemp/', GpsFlowTempResource())

```

Na última linha do código da Listagem A.1 um recurso é adicionado informando o nome do recurso e uma instancia de uma classe. Essa classe, no código “GpsFlowTempResource” tem como função implementar o tratamento de requisições para os métodos POST, GET, PUT e DELETE (Listagem 5.2), herdando características da classe Resource.

Listagem 5.2 – Código do tratador de requisições

```

1     from coapthon.server.coap import CoAP
2     from coapthon.resources.resource import Resource
3     from dbhandler import DataBase
4
5     class GpsFlowTempResource(Resource):
6         def __init__(self, name="gpsflowtemp", coap_server=None):
7             super(GpsFlowTempResource, self).__init__(name, coap_server,
8                                                         visible=True,
9                                                         observable=True,
10                                                         allow_children=True)
11
12             self.payload = None
13             self.dbhandler = DataBase()
14             self.dbhandler.connect_db()
15
16         def render_GET(self, request):
17             return self
18
19         def render_PUT(self, request):
20             return self
21
22         def render_POST(self, request):
23             res = self.init_resource(request, GpsFlowTempResource())
24             res.location_query = request.uri_query
25             res.payload = request.payload
26             print("\n"+res.payload)
27             self.dbhandler.on_message(request.payload)
28             return res

```

```

27     def render_DELETE(self, request):
28         return True

```

A única requisição esperada no escopo deste trabalho é POST (Listagem 5.2, linhas 19 até 25), com o nome do sensor, seu MAC Address, a temperatura, o fluxo, a localização e um timestamp (que estão encapsulados no “request”). Ao receber uma requisição POST, um método de persistência é invocado a fim de salvar esses dados no banco de dados (Listagem 5.2, linha 24).

5.4.2 Desenvolvimento da parte cliente

5.4.2.1 Cliente CoAP com DTLS

O código fonte da parte cliente do CoAP junto ao DTLS foi criada com base no método “test_ok_with_handshake_on_send” presente no arquivo “test_secure.py”. Assim como na parte do servidor, na parte do cliente há a configuração dos certificados, inicialmente, seguida pela configuração do socket utilizando DTLS (Listagem 5.3). Diferentemente do servidor, a configuração do socket cliente envolve apenas o certificado da autoridade certificadora e na parte do cliente que é definido o conjunto de algoritmos criptográficos (cipher suite). Em seguida um cliente é instanciado pelo construtor da classe HelperClient, passando o endereço do servidor e o socket criado.

Listagem 5.3 – Inicialização do cliente

```

1  class coapDtlsClient(object):
2
3      def __init__(self, host, port):
4          #Set Up certificates
5          self.pem = self._setUpPems()
6
7          # Set up a client side DTLS socket
8          _sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
9          _sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
10         _sock = wrap_client(_sock,
11                             cert_reqs=ssl.CERT_REQUIRED,
12                             ca_certs=self.pem['CA_CERT'],
13                             ciphers="ECDHE+AESGCM",
14                             do_handshake_on_connect=True)
15
16         # Connect the CoAP client to the newly created socket
17         self.server_address = (host, port)
18         self.client = HelperClient(self.server_address,
19                                   sock=_sock,

```

```

20         cb_ignore_read_exception=self._cb_ignore_read_exception,
21         cb_ignore_write_exception=self._cb_ignore_write_exception)

```

Inicialmente, no arquivo exemplo “test_security.py”, o parâmetro “ciphers” do método `wrap_client` estava definido da seguinte forma: `ciphers=“RSA”`. Essa definição com uso único do RSA causou dúvidas sobre a definição efetiva do Cipher Suite, descrito na seção 3.2.3, pois faltavam as definições dos algoritmos de criptografia simétrica e HMAC, considerando que o RSA fosse utilizado para troca de chaves e autenticação. Inclusive, sem a definição explícita do algoritmo de criptografia simétrica, não se tinha certeza de que a mensagem era cifrada.

Por causa dessa incerteza, o Wireshark foi utilizado para capturar pacotes do protocolo DTLS e verificar a definição do Cipher Suite durante o *handshake* - apresentado na seção 3.2.2 - e verificar também se mensagens eram transmitidas com criptografia.

Figura 19 – Cipher Suite resultante da atribuição `ciphers=“RSA”` no código fonte

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.25.7	35.231.29.66	DTLSv1.2	182	Client Hello
2	0.140802429	35.231.29.66	192.168.25.7	DTLSv1.2	86	Hello Verify Request
3	0.140948945	192.168.25.7	35.231.29.66	DTLSv1.2	198	Client Hello
4	0.279950047	35.231.29.66	192.168.25.7	DTLSv1.2	121	Server Hello
5	0.282426225	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
6	0.282728836	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
7	0.282730647	35.231.29.66	192.168.25.7	DTLSv1.2	117	Certificate (Reassembled)
8	0.283047646	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
9	0.283120137	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
10	0.283295911	35.231.29.66	192.168.25.7	DTLSv1.2	67	Server Hello Done
11	0.283467328	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
12	0.284032741	192.168.25.7	35.231.29.66	DTLSv1.2	272	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
15	1.280845150	192.168.25.7	35.231.29.66	DTLSv1.2	197	Client Key Exchange
16	1.281079630	192.168.25.7	35.231.29.66	DTLSv1.2	56	Change Cipher Spec
17	1.281122947	192.168.25.7	35.231.29.66	DTLSv1.2	193	Encrypted Handshake Message
18	1.418754533	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate [Reassembly error, protocol DTLS: New fragment overlaps old data...
19	1.418805920	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate [Reassembly error, protocol DTLS: New fragment overlaps old data...
20	1.419043077	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate [Reassembly error, protocol DTLS: New fragment overlaps old data...

```

Cookie: 4ef5c2051466ea87e87d35f44921a88b
Cipher Suites Length: 28
- Cipher Suites (14 suites)
  Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
  Cipher Suite: TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (0x0084)
  Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
  Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
  Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
  Cipher Suite: TLS_RSA_WITH_SEED_CBC_SHA (0x0096)
  Cipher Suite: TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (0x0041)
  Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)

```

Fonte: Elaborada pelo autor

A Figura 19 mostra através de uma captura, na linha selecionada na parte inferior em cor preta, quais Cipher Suites eram indicados pelo cliente com a atribuição de “RSA” ao parâmetro “ciphers” no código fonte. Dessa forma, pode-se concluir que as mensagens eram criptografadas, pois o estabelecimento de um Cipher Suite com algoritmo de criptografia simétrica ocorria. Todas as opções de Cipher Suites envolvem o “RSA” e têm opções de criptografia simétrica. O “RSA” foi usado obrigatoriamente nesta biblioteca, com o uso dos certificados X.509, para realização da troca de chaves e autenticação.

O RSA não é o melhor algoritmo para a troca de chaves, devido à ausência de *forward secrecy*, isto é, o RSA depende da chave privada para a comunicação segura. Quando o Cipher Suite não suporta *forward secrecy*, algum atacante que obtém a chave privada do servidor consegue decifrar todas as mensagens transmitidas anteriormente, caso coletadas (Ristic, 2016).

Seguindo as boas práticas descritas na referência (Ristic, 2016), o ECDHE (Diffie-Hellman Efêmero de Curvas Elípticas) tem a propriedade de *forward secrecy*, por isso, o cliente deveria utilizar um Cipher Suite envolvendo esse algoritmo e não utilizar troca de chaves com o RSA.

Todavia, não se sabia qual string deveria ser atribuída a “ciphers” para a obtenção de um Cipher Suite com ECDHE. Após uma pesquisa nos arquivos de código importados das bibliotecas pertencentes ao python 2.7 - no arquivo “ssl.py” - foram encontradas as definições de strings para Cipher Suites descritas na Listagem 5.4.

Listagem 5.4 – Cipher Suites no código ssl.py

```

1  _DEFAULT_CIPHERS = (
2      'ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+HIGH:'
3      'DH+HIGH:ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+HIGH:RSA+3DES:!aNULL:'
4      '!eNULL:!MD5'
5  )
6
7  _RESTRICTED_SERVER_CIPHERS = (
8      'ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+HIGH:'
9      'DH+HIGH:ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+HIGH:RSA+3DES:!aNULL:'
10     '!eNULL:!MD5:!DSS:!RC4'
11 )

```

Com base nessas strings definidas no arquivo “ssl.py”, a atribuição para o parâmetro “ciphers” foi realizada com a string “ECDHE+AESGCM”, esperando-se uma definição de cipher suite com o Diffie-Hellman-Efêmero de Curvas Elípticas para troca de chaves e com AES em modo GCM (Galois/Counter Mode) para criptografia simétrica autenticada. A Figura 20 mostra, na linha selecionada na parte inferior em cor preta, a obtenção da configuração desejada.

Figura 20 – Captura *handshake* DTLS (Client Hello)

No.	Time	Source	Destination	Protocol	Length	Info
2458	1.843288215	192.168.25.7	35.231.29.66	DTLSv1.2	204	Client Hello
2471	1.983844914	35.231.29.66	192.168.25.7	DTLSv1.2	86	Hello Verify Request
2472	1.983974885	192.168.25.7	35.231.29.66	DTLSv1.2	220	Client Hello
2499	2.126111983	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2500	2.126817315	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2501	2.126818209	35.231.29.66	192.168.25.7	DTLSv1.2	117	Certificate (Reassembled)
2502	2.126818854	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2503	2.126820795	35.231.29.66	192.168.25.7	DTLSv1.2	268	Server Key Exchange
2504	2.127029006	35.231.29.66	192.168.25.7	DTLSv1.2	67	Server Hello Done
2505	2.127031251	35.231.29.66	192.168.25.7	DTLSv1.2	129	Server Hello
2506	2.127809045	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2507	2.127833195	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2508	2.128563509	192.168.25.7	35.231.29.66	DTLSv1.2	208	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Mess...
2516	2.267908427	35.231.29.66	192.168.25.7	DTLSv1.2	56	Change Cipher Spec
2518	2.269026802	35.231.29.66	192.168.25.7	DTLSv1.2	233	New Session Ticket
2520	2.269031361	35.231.29.66	192.168.25.7	DTLSv1.2	103	Encrypted Handshake Message
2530	3.127679319	192.168.25.7	35.231.29.66	DTLSv1.2	133	Client Key Exchange
2531	3.127748083	192.168.25.7	35.231.29.66	DTLSv1.2	56	Change Cipher Spec

Cookie: 1918edcfff590be3b8ea5b4c643c7c7a7

Cipher Suites Length: 10
 - Cipher Suites (5 suites)

Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
 Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
 Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
 Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)

Compression Methods Length: 1
 - Compression Methods (1 method)

Extensions Length: 85
 - Extension: ec_point_formats (len=4)
 - Extension: supported_groups (len=28)

Fonte: Elaborada pelo autor

A Figura 20 mostra a sugestão de possíveis Cipher Suites por parte do cliente e verifica-se que todas elas estão de acordo com a configuração do parâmetro “ciphers” no código fonte do cliente.

Figura 21 – Captura *handshake* DTLS (Server Hello)

No.	Time	Source	Destination	Protocol	Length	Info
2458	1.843288215	192.168.25.7	35.231.29.66	DTLSv1.2	204	Client Hello
2471	1.983844914	35.231.29.66	192.168.25.7	DTLSv1.2	86	Hello Verify Request
2472	1.983974885	192.168.25.7	35.231.29.66	DTLSv1.2	220	Client Hello
2499	2.126111983	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2500	2.126817315	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2501	2.126818209	35.231.29.66	192.168.25.7	DTLSv1.2	117	Certificate (Reassembled)
2502	2.126818854	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2503	2.126820795	35.231.29.66	192.168.25.7	DTLSv1.2	268	Server Key Exchange
2504	2.127029006	35.231.29.66	192.168.25.7	DTLSv1.2	67	Server Hello Done
2505	2.127031251	35.231.29.66	192.168.25.7	DTLSv1.2	129	Server Hello
2506	2.127809045	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2507	2.127833195	35.231.29.66	192.168.25.7	DTLSv1.2	270	Certificate (Fragment)
2508	2.128563509	192.168.25.7	35.231.29.66	DTLSv1.2	208	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Mess...
2516	2.267908427	35.231.29.66	192.168.25.7	DTLSv1.2	56	Change Cipher Spec
2518	2.269026802	35.231.29.66	192.168.25.7	DTLSv1.2	233	New Session Ticket
2520	2.269031361	35.231.29.66	192.168.25.7	DTLSv1.2	103	Encrypted Handshake Message
2530	3.127679319	192.168.25.7	35.231.29.66	DTLSv1.2	133	Client Key Exchange
2531	3.127748083	192.168.25.7	35.231.29.66	DTLSv1.2	56	Change Cipher Spec

Fragment Offset: 0
 Fragment Length: 62
 Version: DTLS 1.2 (0xfefd)
 - Random: 1a92eb6f005f72202e2bd4e0c782757904c431f7901faa5...
 Session ID Length: 0

Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
 Compression Method: null (0)
 Extensions Length: 22
 - Extension: renegotiation_info (len=1)
 - Extension: ec_point_formats (len=4)
 - Extension: SessionTicket TLS (len=0)
 - Extension: heartbeat (len=1)

Fonte: Elaborada pelo autor

A Figura 21 mostra, na linha selecionada na parte inferior em cor preta, que o servidor escolheu a primeira opção de Cipher Suite sugerida pelo cliente: **TLS_ECDHE_RSA_WITH_**

5.4.2.2 Código principal do cliente

O código da Listagem 5.5 representa o funcionamento da Raspberry Pi, que com uma instância de “coapDtlsClient” e com uma instância do sensor de temperatura, lê a temperatura do ambiente, lê o MAC address da Raspberry Pi e lê um timestamp

referente à hora atual da Raspberry Pi. Em seguida formata os dados seguindo a função “mountPayload” e envia esses dados ao recurso “gpsflowtemp”, presente no servidor CoAP.

Listagem 5.5 – Parte do código principal do cliente

```

1 clienteDtls = coapDtlsClient("35.231.29.66", 5684).client
2 temperatureSensor = TempSensor(temp_file);
3 try:
4     while True:
5         temperature = str(temperatureSensor.read_temp());
6         macAddress = str(get_mac())
7         timestamp = str(time.time())
8         payload = mountPayload("SENSOR_01", macAddress, temperature, str(0),
9                               "NaMinhaCasa", timestamp)
10        response = clienteDtls.post("gpsflowtemp", payload)
11        print response.pretty_print()
12        time.sleep(60)

```

Nota-se a ausência do fluxo e da localização a partir da leitura de sensores, pois eles não puderam ser obtidos para os testes.

5.4.3 Mensagens de Debug

Tanto na implementação do cliente quanto na implementação do servidor, a biblioteca CoAPthon cria o arquivo “logging.conf” e usa esse arquivo como base de configuração para o debugger do python (Listagem 5.6).

Listagem 5.6 – Configuração do debugger

```

1 if not os.path.isfile("logging.conf"):
2     create_logging()
3
4 logger = logging.getLogger(__name__)
5 logging.config.fileConfig("logging.conf", disable_existing_loggers=False)

```

As configurações possíveis estão disponíveis na documentação da linguagem Python, na seção com título “Logging configuration”.

5.4.4 Testes

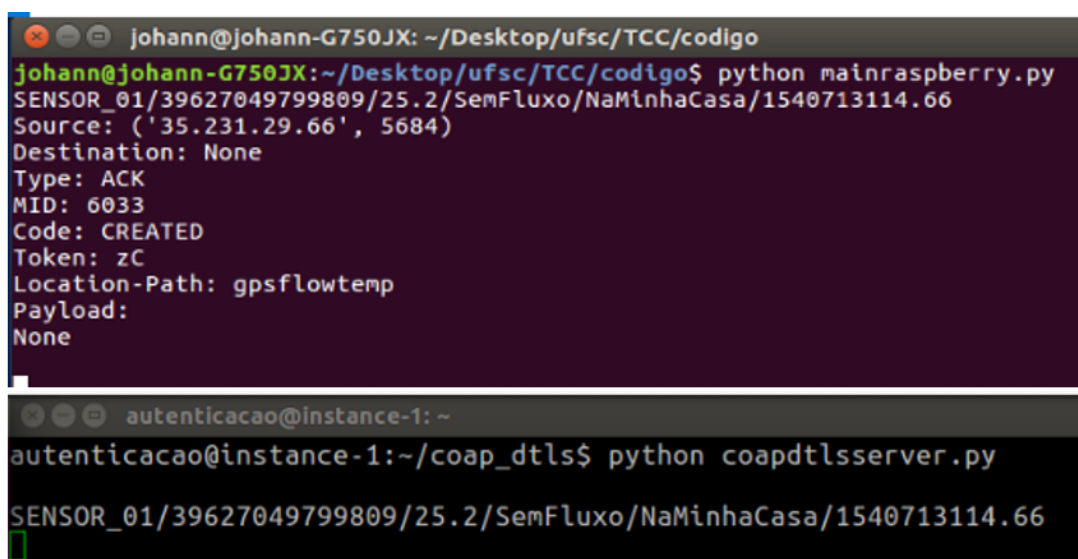
Após a implantação do servidor, descrito na seção 5.4.1, em uma cloud da Google, foram realizados testes para verificação da corretude dos códigos desenvolvidos.

5.4.4.1 Teste inicial

O primeiro teste foi realizado com um notebook executando o código do cliente, que é executado pela Raspberry Pi na solução definitiva, e o código do servidor na cloud da Google. O objetivo do primeiro teste foi verificar que a string contendo as informações enviadas pelo cliente são decifradas corretamente no servidor (Figura 22). Na primeira linha do terminal mostrado na parte superior da Figura 22 está a string enviada do cliente ao servidor, com informações em sequência separadas por “/”.

A primeira informação, “SENSOR_01” representa o nome da Raspberry Pi que está enviando os dados. Em seguida, o número “39627049799809” representa o MAC address da Raspberry Pi. As próximas três informações representam respectivamente a temperatura em Grau Celsius, o fluxo e a localização lidos pelos sensores. Por último, é enviado um timestamp considerando o relógio do sistema operacional da Raspberry Pi. Nas linhas seguinte do mesmo terminal, estão os campos do cabeçalho do CoAP referentes ao Acknowledgement enviado pelo servidor ao receber as informações da primeira linha.

Figura 22 – Interação entre cliente e servidor utilizando CoAP e DTLS



```
johann@johann-G750JX: ~/Desktop/ufsc/TCC/codigo
johann@johann-G750JX:~/Desktop/ufsc/TCC/codigo$ python mainraspberrypi.py
SENSOR_01/39627049799809/25.2/SemFluxo/NaMinhaCasa/1540713114.66
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 6033
Code: CREATED
Token: zC
Location-Path: gpsflowtemp
Payload:
None

autenticacao@instance-1: ~
autenticacao@instance-1:~/coap_dtls$ python coapdtlsserver.py
SENSOR_01/39627049799809/25.2/SemFluxo/NaMinhaCasa/1540713114.66
```

Fonte: Elaborada pelo autor

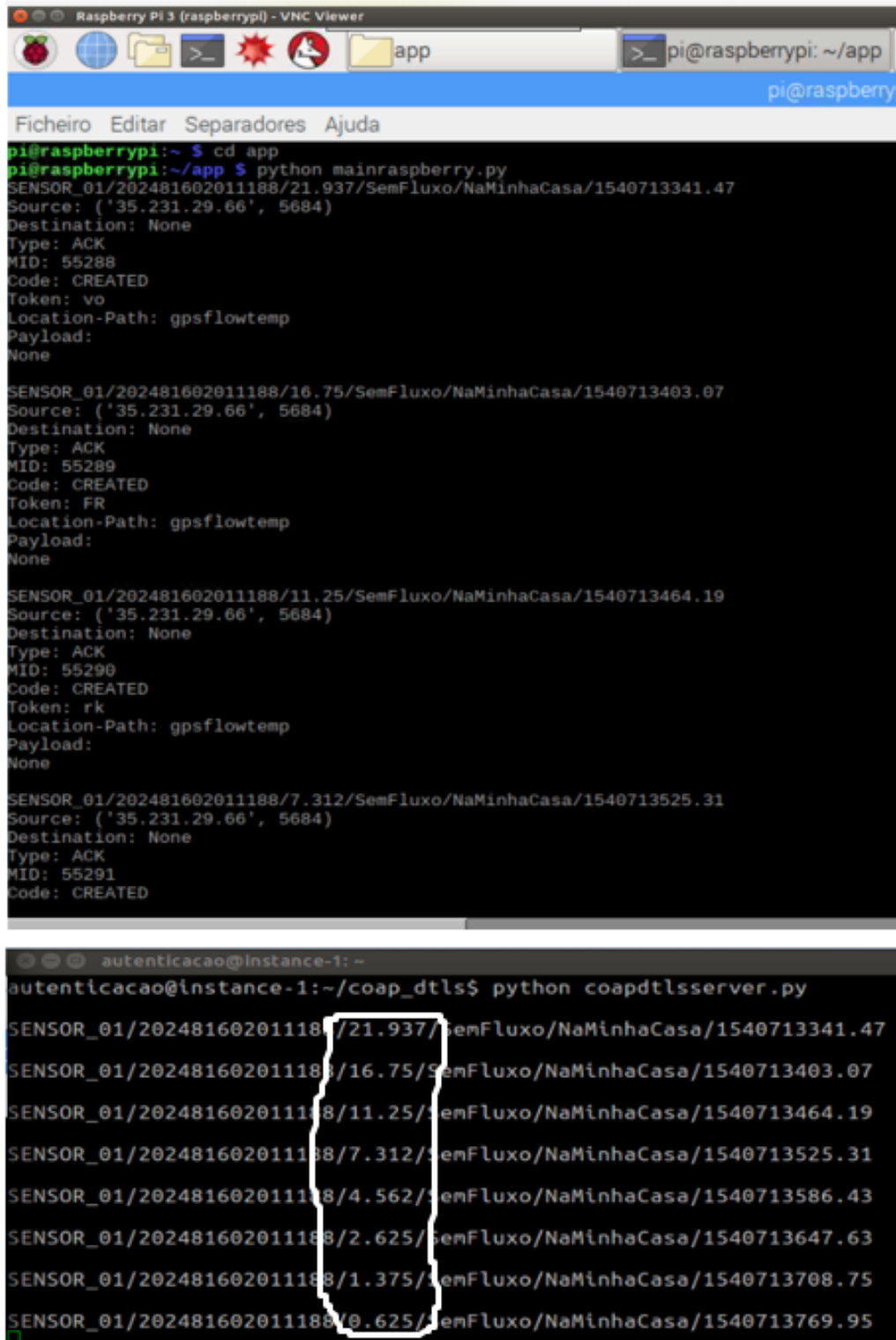
A Figura 22 mostra na janela superior a execução do cliente, executado em um notebook, com uma temperatura fictícia de 25.2°C e na janela inferior a execução do servidor. Nota-se que a mensagem enviada pelo cliente é recebida com sucesso no servidor e o ACK relativo ao protocolo CoAP é recebido pelo cliente como indicação de sucesso na comunicação.

5.4.4.2 Teste com Raspberry Pi e sensor de temperatura

A Figura 23 mostra o teste do código desenvolvido em execução na Raspberry Pi, para verificar o funcionamento do código de leitura da temperatura e identificar possíveis

ausências de bibliotecas python necessárias. A fim de controlar a Raspberry Pi remotamente, foi utilizado o programa VNC Viewer.

Figura 23 – Interação entre Raspberry Pi e servidor utilizando CoAP e DTLS



```
Raspberry Pi 3 (raspberrypi) - VNC Viewer
pi@raspberrypi: ~/app
Ficheiro Editar Separadores Ajuda
pi@raspberrypi:~$ cd app
pi@raspberrypi:~/app$ python mainraspberry.py
SENSOR_01/202481602011188/21.937/SemFluxo/NaMinhaCasa/1540713341.47
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 55288
Code: CREATED
Token: vo
Location-Path: gpsflowtemp
Payload:
None

SENSOR_01/202481602011188/16.75/SemFluxo/NaMinhaCasa/1540713403.07
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 55289
Code: CREATED
Token: FR
Location-Path: gpsflowtemp
Payload:
None

SENSOR_01/202481602011188/11.25/SemFluxo/NaMinhaCasa/1540713464.19
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 55290
Code: CREATED
Token: rk
Location-Path: gpsflowtemp
Payload:
None

SENSOR_01/202481602011188/7.312/SemFluxo/NaMinhaCasa/1540713525.31
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 55291
Code: CREATED

autenticacao@Instance-1: ~
autenticacao@instance-1:~/coap_dtls$ python coapdtlserver.py
SENSOR_01/202481602011188/21.937/SemFluxo/NaMinhaCasa/1540713341.47
SENSOR_01/202481602011188/16.75/SemFluxo/NaMinhaCasa/1540713403.07
SENSOR_01/202481602011188/11.25/SemFluxo/NaMinhaCasa/1540713464.19
SENSOR_01/202481602011188/7.312/SemFluxo/NaMinhaCasa/1540713525.31
SENSOR_01/202481602011188/4.562/SemFluxo/NaMinhaCasa/1540713586.43
SENSOR_01/202481602011188/2.625/SemFluxo/NaMinhaCasa/1540713647.63
SENSOR_01/202481602011188/1.375/SemFluxo/NaMinhaCasa/1540713708.75
SENSOR_01/202481602011188/0.625/SemFluxo/NaMinhaCasa/1540713769.95
```

Fonte: Elaborada pelo autor

O sensor de temperatura foi colocado sobre uma bolsa de gelo e, como mostra a

Figura 23, os envios da Raspberry Pi ao servidor mostram a queda de temperatura.

6 Conclusão

6.1 Principais Contribuições

Com o aumento do uso de dispositivos IoT, o atual trabalho traz contribuições para o ambiente de IoT de diversas maneiras. Primeiramente, o trabalho contém um estudo sobre dois protocolos relevantes, CoAP e DTLS, que podem facilitar a comunicação e a segurança dos dispositivos IoT.

Além do estudo teórico, o trabalho apresenta uma solução prática de IoT utilizando a combinação CoAP e DTLS, o que ainda não se encontra em artigos científicos e trabalhos acadêmicos com frequência. Ainda no escopo do desenvolvimento, a biblioteca CoAPthon possuía apenas um teste envolvendo DTLS, com códigos referentes ao cliente e ao servidor acoplados em um arquivo. Este trabalho demonstrou como criar em classes separadas o cliente e o servidor, utilizando CoAP e DTLS a partir da CoAPthon, com informações pertinentes de cada um.

O conteúdo deste trabalho auxilia também futuros trabalhos que utilizem a CoAPthon, já que a biblioteca não possui documentação completa. Neste trabalho foi desenvolvido um exemplo abordando também aspectos de configuração de segurança.

6.2 Sugestões de trabalhos futuros

Trabalhos futuros podem tratar de aspectos ou realizar análises não cobertos neste trabalho, como por exemplo a integração de hardware e software do sistema desenvolvido neste trabalho com os sensores de fluxo e de localização. Também seria interessante um trabalho que adaptasse os códigos utilizados para dispositivos mais limitados do que uma Raspberry Pi, afinal, eles são o alvo principal do desenvolvimento de protocolos como DTLS e CoAP. Ainda no contexto de dispositivos limitados, pode ser realizado um estudo acerca da otimização entre a escolha de algoritmos de segurança e o consumo energético, espaço e tempo de processamento necessários.

Referências

- BANERJEE, U. et al. An energy-efficient reconfigurable dtls cryptographic engine for end-to-end security in iot applications. In: *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018. p. 42–44. ISSN 2376-8606. Citado 3 vezes nas páginas 39, 40 e 42.
- BARR, M. *Introduction to Pulse Width Modulation (PWM)*. 2001. Acessado em 4 de outubro de 2018. Disponível em: <<https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation>>. Citado na página 45.
- BORMANN, C. *The Constrained Application Protocol (CoAP)*. 2014. Acessado em 3 de outubro de 2018. Disponível em: <<https://tools.ietf.org/html/rfc7252>>. Citado 3 vezes nas páginas 29, 48 e 49.
- BORMANN, C. *Block-Wise Transfers in the Constrained Application Protocol (CoAP)*. 2016. Acessado em 16 de Junho de 2018. Disponível em: <<https://tools.ietf.org/html/rfc7959>>. Citado na página 31.
- BORMANN, C.; CASTELLANI, A. P.; SHELBY, Z. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, v. 16, n. 2, p. 62–67, March 2012. ISSN 1089-7801. Citado 4 vezes nas páginas 27, 28, 30 e 31.
- CHRIS. *Using A Flow Sensor With Arduino*. 2015. Acessado em 1 de dezembro de 2018. Disponível em: <<https://www.bc-robotics.com/tutorials/using-a-flow-sensor-with-arduino/>>. Citado na página 45.
- DESCONHECIDO. *Raspberry Pi 3 Model B Motherboard*. 2016. Acessado em 16 de outubro de 2018. Disponível em: <<https://www.amazon.com/Raspberry-Pi-RASPBERRYPI3-MODB-1GB-Model-Motherboard/dp/B01CD5VC92>>. Citado na página 47.
- DIERKS, T. *The Transport Layer Security (TLS) Protocol Version 1.2*. 2008. Acessado em 23 de junho de 2018. Disponível em: <<https://tools.ietf.org/html/rfc5246>>. Citado na página 33.
- FIELDING, R. T. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doctoral dissertation) — University of California, Irvine, 2000. Disponível em: <<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>>. Citado na página 31.
- GONÇALVES, E. *O que é o Raspberry Pi 3 (rpi3)*. 2018. Acessado em 17 de outubro de 2018. Disponível em: <<https://www.mundotibrasil.com.br/o-que-e-o-raspberry-pi-3-rpi3/>>. Citado na página 47.
- GRANJAL, J.; MONTEIRO, E.; SILVA, J. S. Security for the internet of things: A survey of existing protocols and open research issues. *IEEE Communications Surveys Tutorials*, v. 17, n. 3, p. 1294–1312, thirdquarter 2015. ISSN 1553-877X. Citado 2 vezes nas páginas 19 e 23.

GROKHOTOKOV, I.; MOLINARI, M. *CoAP protocol for ESP-12E*. 2018. Acessado em 13 de agosto de 2018. Disponível em: <<https://github.com/esp8266/Arduino>>. Citado na página 48.

KAMALUDIN, K. H.; ISMAIL, W. Water quality monitoring with internet of things (iot). In: *2017 IEEE Conference on Systems, Process and Control (ICSPC)*, 2017. p. 18–23. Citado 3 vezes nas páginas 40, 41 e 42.

KOTHMAYR, T. et al. Dtls based security and two-way authentication for the internet of things. *Ad Hoc Networks*, v. 11, n. 8, p. 2710 – 2723, 2013. ISSN 1570-8705. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1570870513001029>>. Citado 2 vezes nas páginas 34 e 36.

MALAN, C. *ESP8266 NodeMCU Placa Módulo de Desenvolvimento de WiFi - PRETO 1*. 2017. Acessado em 1 de dezembro de 2018. Disponível em: <<https://medium.com/@cilliemalan/installing-nodemcu-drivers-on-windows-d9bffd52>>. Citado na página 46.

OLSSON, J. *6LoWPAN demystified*. 2014. Acessado em 5 de Junho de 2018. Disponível em: <<http://www.ti.com/lit/wp/swry013/swry013.pdf>>. Citado 2 vezes nas páginas 25 e 26.

PORCIÚNCULA, C. B. da et al. Constrained application protocol (coap) no arduino uno r3: Uma análise prática. In: , 2018. v. 5. Disponível em: <<http://ojs.sbc.org.br/index.php/wpietf/article/view/3212>>. Citado 2 vezes nas páginas 39 e 42.

POSTCAPES. *What Is The Internet of Things?* 2015. Acessado em 4 de Junho de 2018. Disponível em: <<https://www.postscapes.com/what-exactly-is-the-internet-of-things-infographic/>>. Citado 2 vezes nas páginas 23 e 24.

RAZA, S. et al. S3k: Scalable security with symmetric keys x2014;dtls key establishment for the internet of things. *IEEE Transactions on Automation Science and Engineering*, v. 13, n. 3, p. 1270–1280, July 2016. ISSN 1545-5955. Nenhuma citação no texto.

RESCORLA, E. *Datagram Transport Layer Security Version 1.2*. 2012. Acessado em 19 de Junho de 2018. Disponível em: <<https://tools.ietf.org/html/rfc6347>>. Citado 2 vezes nas páginas 34 e 35.

RISTIC, I. *OpenSSL Cookbook: A Guide to the Most Frequently Used OpenSSL Features and Commands*. Feisty Duck, 2016. ISBN 9781907117053. Disponível em: <<https://www.feistyduck.com/books/openssl-cookbook/>>. Citado 2 vezes nas páginas 53 e 54.

SINGH, P. *NodeMCU Pinout*. 2016. Acessado em 4 de outubro de 2018. Disponível em: <<https://iotbytes.wordpress.com/nodemcu-pinout/>>. Citado na página 47.

TANGANELLI, G. *CoAPthon is a python library to the CoAP protocol aligned with the RFC*. 2018. Acessado em 16 de outubro de 2018. Disponível em: <<https://github.com/Tanganelli/CoAPthon>>. Citado na página 49.

- TANGANELLI, G.; VALLATI, C.; MINGOZZI, E. Coapthon: Easy development of coap-based iot applications with python. In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015. p. 63–68. Citado na página 49.
- TAVARES, S. A. C. et al. Telemetry for domestic water consumption based on iot and open standards. In: *2018 Workshop on Metrology for Industry 4.0 and IoT*, 2018. p. 1–6. Citado 2 vezes nas páginas 41 e 42.
- THOMSEN, A. *Medindo temperatura debaixo d'água com DS18B20*. 2015. Acessado em 3 de outubro de 2018. Disponível em: <<https://www.filipeflop.com/blog/sensor-de-temperatura-ds18b20-arduino/>>. Citado na página 45.
- WALLACE; RICHARDSON, M. *Getting Started With Raspberry Pi: An Introduction to the Fastest-Selling Computer in the World*. 3rd. ed. USA: Maker Media, Inc, 2016. ISBN 1680452460, 9781680452464. Citado na página 47.
- ZHENG, J.; LEE, M. J. A comprehensive performance study of iee 802.15. 4. *Sensor network operations*, v. 4, p. 218–237, 2006. Nenhuma citação no texto.
- ZHOU, J. et al. Security and privacy for cloud-based iot: Challenges. *IEEE Communications Magazine*, v. 55, n. 1, p. 26–33, January 2017. ISSN 0163-6804. Citado na página 23.

Apêndices

APÊNDICE A – Código Desenvolvido

Neste apêndice estão os códigos desenvolvidos neste trabalho. Vale ressaltar que para que ele seja executado com sucesso precisa da biblioteca CoAPthon instalada no computador de execução.

Listagem A.1 – coapdtlsclient.py

```

1 # System
2 import socket
3 import random
4 import threading
5 import unittest
6 import tempfile
7 import os
8 import time
9
10 # Dtls
11 import ssl
12 from dtls.wrapper import wrap_server, wrap_client
13
14 # CoAPthon
15 from coapthon import defines
16 from coapthon.client.helperclient import HelperClient
17 from coapthon.messages.option import Option
18 from coapthon.messages.request import Request
19 from coapthon.messages.response import Response
20
21 # Logging
22 #from logging import basicConfig, DEBUG, getLogger, root, Filter
23 #basicConfig(level=DEBUG, format="%(asctime)s - %(threadName)-30s - %(name)s -
    %(levelname)s - %(message)s")
24
25
26 CA_CERT_VALID = """-----BEGIN CERTIFICATE-----
27 MIICczCCAXQCCQcwSKaN4J3cTANBgkqhkiG9w0BAQUFADBKMQswCQYDVQQGEwJV
28 UzETMBEGA1UECBMKV2FzaGluZ3RvbjETMBEGA1UEChMKUmf5IENBIEluYzERMA8G
29 A1UEAxMIUmF5Q0FJbMwHhcNMTQwMTE2MjEwMjUwWWhcNMjEwMjUwMjUwWWhc
30 MQswCQYDVQQGEwJVUzETMBEGA1UECBMKV2FzaGluZ3RvbjETMBEGA1UEChMKUmf5
31 IENBIEluYzERMA8GA1UEAxMIUmF5Q0FJbMwHhcZ8wDQYJKoZIhvcNAQEBBQADgYOA
32 MIGJAoGBAN/UYXt4uq+YdTDnm7WPCu+0B50kJXWU3sSS+WAAhr3BHh7qa7UTiRXY

```

```
33 yGYysgvtwriETAZRckzd+hdblNRUWXGJdRvtyx94nLpPpI8p4djBrJ5IMPqK5SgW
34 ZP4XTWs694VtUBAvHCX+Ly+t005Rw3NmqxY1MakooqU9t+wLOHOTAgMBAAEwDQYJ
35 KoZIHvcNAQEFBQADgYEANemjvYCJrTc/6im0DmDC6AW8KrLG0xj31HWpq1d09LG7
36 m1VFgbVtbcuCZgA78kxgw1vN6kBBLEsAJC8gkg++A0/w3a4oP+U9txAr9KRg6IGA
37 FiUohuWbjKBnQEpcEOECgrymooF3ayzke/vf3wcMYy153uC+H4t96Yc5T066c4o=
38 -----END CERTIFICATE-----
39 ""
40
41 CA_CERT_WRONG = ""-----BEGIN CERTIFICATE-----
42 MIE6jCCBF0gAwIBAgIDEIGKMAOGCSqSIB3DQEBBQUAME4xCzAJBgNVBAYTA1VT
43 MRAwDgYDVQQKEwdFcXVpZmF4MS0wKwYDVQQLEyRfCXVpZmF4IFN1Y3VyZSBDZXJ0
44 aWZpY2F0ZSBBDXR0b3JpdHkwHhcNMTAwNDAxMjMwMDE0WhcNMTUwNzAzMDQ1MDAw
45 WjCBjzEpMCCGA1UEBRMgMmc4YU81d0kxYktKMLpENTg4VXNMDkRlM2dUYmc4RFRUx
46 CzAJBgNVBAYTA1VTMRMwEQYDVQQIEWpDYWxpZm9ybmlhMRIwEAYDVQQHEw1TdW5u
47 eXZhbGUxZDASBgNVBAoTC1lhaG9vICBjbMUMRYWFAyDVQQDEw13d3cueWFob28u
48 Y29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA6ZM1jHCkL8r1EKse
49 1riTTxyC3WvYQ5m34TlFK7dK4QFI/HPttKGqQm3aVB1Fqi0aiTxe4YQMbd++jnKt
50 djxcpi7sJlFxmZs4umr1eGo2KgTgSBAJyhxo23k+VpK1SprdPyM3yEfQVdV7JWC
51 4Y71CE2nE6+GbsIuhk/to+jJM07jXx/430jvo8vhNPL6GvWe/D60bbnxS72ynLSd
52 mLtaltky0vZEZiXbbFKGIaYymCgh89FGVvBkUbGM/Wb5Voiz7ttQLLxKOYRj8Mdk
53 TZtzPkM9scIFG1naECPvCwx0NyMyxY3nF0djUKJ79twanmfCclX2ZO/rk1Cpi0uw
54 lrrr/QIDAQABo4ICDjCCAgowDgYDVR0PAQH/BAQDAgTwMBOGA1UdDgQWBBSmrfKs
55 68m+dDUSf+S7xJrQ/FXAlzA6BgNVHR8EMzAxMC+gLaArhilodHRwOi8vY3JsLmdl
56 b3RydXN0LmNvbS9jcmxzL3N1Y3VyZW50b28uY29tMDE0WhcNMTUwNzAzMDQ1MDAw
57 d3cueWFob28uY29tgg15YWhvby5jb22CDHVzLnlhaG9vLmNvbYIMa3IueWFob28u
58 Y29tggx1ay55YWhvby5jb22CDG11LnlhaG9vLmNvbYIMZnIueWFob28uY29tggxp
59 bi55YWhvby5jb22CDGNhLnlhaG9vLmNvbYIMYnIueWFob28uY29tggxkZS55YWhv
60 by5jb22CDGVzLnlhaG9vLmNvbYIMbXgueWFob28uY29tggxpdC55YWhvby5jb22C
61 DHNnLnlhaG9vLmNvbYIMaWQueWFob28uY29tggxwC55YWhvby5jb22CDHFjLnlh
62 aG9vLmNvbYIMdHcueWFob28uY29tggxoay55YWhvby5jb22CDGNuLnlhaG9vLmNv
63 bYIMYXUueWFob28uY29tggxhci55YWhvby5jb22CDHZuLnlhaG9vLmNvbTAfBgNV
64 HSMEDAwgBRI5mj5K9KylddH2CMgEE8zmJCf1DAdBgNVHSUEFjAUBggrBgEFBQcD
65 AQYIKwYBBQUHAwIwDQYJKoZIhvcNAQEFBQADgYEAp9WOMtcDMM5T0yfPecGv5QhH
66 RJZRzgeMPZitLksr1JxxicJrdgv82Nwq1bw8aMurj47ijrtaTEWxaCCy00yXodD
67 zoRJVNoYIvY1arYZf5zv9VZjN5IOHqUc39mNMe9XdZtbkWE+K6yVh60imKLbizna
68 inu9YTrN/4P/w6KzHho=
69 -----END CERTIFICATE-----
70 ""
71
72 KEY_CERT_VALID = ""-----BEGIN PRIVATE KEY-----
73 MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBANjL+g7MpTEB40Vo
74 2pxWbx33YwgXQ6QbnLg1QyKlrH6DEEotyDRWI/ZftvWbjGUh0zUGhQaLzF3ZNgdM
```



```

75 VkJF5j0wCgRdwPon1ct5wJUg6GCWvfi4B/HlQrWg8JDaWoGuDcTqLh6KYfDdWTlWC
76 Bq3p0W14gVe3d12R8Bxu9PCK8jrvAgMBAAECgYAQFjqs5HSRiWFS4i/uj99Y6uV3
77 UTqcr8vWQ2WC6aY+EP2hc3o6n/W1L28FFJC7ZGImuiAe1zrH7/k5W2m/HAUM7M9p
78 oBcp7ZVMFU6R00cQWVKCpQRcPNHnn+tVJdRgiHRj9836/u2z3shBxDYgXJIR787V
79 SlBXkCcsi0Clem5ocQJBAPp/OtF4Cpoa0CAnNN+rDjPNGcH571mpSZBMXZVAVCRq
80 vJDdH9SIcb19gKToCF1MUd7CJWbSHKxh49Hr+prBW8cCQqDdjH8EZ4CDYvoJbVX
81 iWfFbh6lPwv8uaj43HoHq4+51mhHvLx08a1AKMSgD2cg7yJYYIpTTAf21gqU3Yt9
82 wJeZaKEA175e4u0o3vkLDs8xRFzGmbKg69SPA11+ap8YAZWaYwUVfVu2MHUHEZA5
83 GyxEB0B6p8pMBeE55WLMw8UHDmNeQJADEWRGjMnm1mAvFUKXFThrdV9oQ2C7nai
84 I1ai87X0+i4kDIUpsP21603ZJjx0K+DS+C4wuzhk4IkugNxcK5SNUQJASxf8E4z5
85 W5rP2XXIohGpDyzI+criUYQ6340vKB9bPsCQ2QooQq1BH0wGA2fy82Kr95E8KhUo
86 zGoP1Dtpzgw0Qg==
87 -----END PRIVATE KEY-----
88 -----BEGIN CERTIFICATE-----
89 MIICDTCCAXYCCQCxc2uXBLZhDjANBgkqhkiG9w0BAQUFADBKMqswCQYDVQQGEwJV
90 UzETMBEGA1UECBMKV2FzaGluZ3Rvb2JEMTBEGA1UEChMKUmF5IENBIEluYzERMA8G
91 A1UEAxMIUmF5Q0FJbmMwHhcNMTQwMTE4MjEwMjUwWhcNMjEwMTE4MjEwMjUwWjBM
92 MQswCQYDVQQGEwJVUzETMBEGA1UECBMKV2FzaGluZ3Rvb2JEMTBEGA1UEChMLUmF5
93 IFNydiBJbmMxEjAQBgNVBAMTCVJheVNYdkluYzCBnzANBgkqhkiG9w0BAQEFAAOB
94 jQAwwYkCgYEA2Mv6DsylMQHjRwjFZvHfdjCBdDpBucuDVDIqWsfomQSi3INFYj
95 91+29ZuMZSHTNqAFBovMXdk2B0xWQXmPTAKBF3A+ifVy3nA1SDoYJa9+LgH8eVct
96 aDwkNpaga4Nx0ouHoph8N1Z0VYIGrek5bXiBV7d3XZHWHG708IryOu8CAwEAATAN
97 BgkqhkiG9w0BAQUFAAOBgQBw0XUTYzfiI0Fi9g4GuyWD2hjET3NtrT4Ccu+Jiivy
98 EvwhzHtVGAPhrV+VCL8sS9uSOZlmfK/ZVraDiFGpJLDMvPP5y5fwq5VGrFuZispG
99 X6bTBq2AIKzGGXxhwPqD8F7su7bmZDnZFRMRk2Bh16rvOmtzx9yHtqC5YJZ2a3JK
100 2g==
101 -----END CERTIFICATE-----
102 ""
103
104
105 class coapDtlsClient(object):
106
107     def __init__(self, host, port, ciphers_arg="ECDHE+AESGCM"):
108         #_logger.info("Called test_ok_with_handshake_on_connect ...")
109
110         # exchange = self._create_test_sequence()
111
112         #Set Up certificates
113         self.pem = self._setUpPems()
114
115         # Set up a client side DTLS socket
116         _sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

```

```
117     _sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
118     tInicio = time.time()
119     _sock = wrap_client(_sock,
120                        cert_reqs=ssl.CERT_REQUIRED,
121                        ca_certs=self.pem['CA_CERT'],
122                        ciphers=ciphers_arg,
123                        do_handshake_on_connect=True)
124     tFim = time.time()
125     print("Instanciacao do socket demorou %f ms na configuracao %s" %
126           (tFim - tInicio, ciphers_arg))
127     # Connect the CoAP client to the newly created socket
128     self.server_address = (host, port)
129     self.client = HelperClient(self.server_address,
130                               sock=_sock,
131                               cb_ignore_read_exception=self._cb_ignore_read_exception,
132                               cb_ignore_write_exception=self._cb_ignore_write_exception)
133
134
135     def _setUpPems(self):
136         def _createPemFile(fname, content):
137             with open(fname, mode='w') as f:
138                 f.write(content)
139             f.close()
140             return fname
141
142         self._tmp_dir = tempfile.mkdtemp()
143
144         pem = dict()
145         pem['SERVER_KEY'] = _createPemFile(os.path.join(self._tmp_dir,
146                                                       'serverkey.pem'), KEY_CERT_VALID)
147         pem['CA_CERT'] = _createPemFile(os.path.join(self._tmp_dir,
148                                                       'ca_cert.pem'), CA_CERT_VALID)
149         pem['CA_CERT_WRONG'] = _createPemFile(os.path.join(self._tmp_dir,
150                                                             'ca_cert_wrong.pem'), CA_CERT_WRONG)
151
152         return pem
153
154     def _cb_ignore_read_exception(self, exception, client):
155         """
156         In the CoAP client read method, different exceptions can arise from
157         the DTLS stack. Depending on the type of exception, a
158         continuation might not be possible, or a logging might be desirable.
```

With this callback both needs can be satisfied.

```

155
156     note: Default behaviour of CoAPthon without DTLS if no
        _cb_ignore_read_exception would be called is with "return False"
157
158     :param exception: What happened inside the DTLS stack
159     :param client: Reference to the running CoAP client
160     :return: True if further processing should be done, False processing
        should be stopped
161     """
162     return False
163
164     def _cb_ignore_write_exception(self, exception, client):
165         """
166         In the CoAP client write method, different exceptions can arise from
        the DTLS stack. Depending on the type of exception, a
167         continuation might not be possible, or a logging might be desirable.
        With this callback both needs can be satisfied.
168
169         note: Default behaviour of CoAPthon without DTLS if no
        _cb_ignore_write_exception would be called is with "return True"
170
171         :param exception: What happened inside the DTLS stack
172         :param client: Reference to the running CoAP client
173         :return: True if further processing should be done, False processing
        should be stopped
174         """
175         return False

```

Listagem A.2 – mainraspberry.py

```

1 from tempsensor import TempSensor
2 from coapdtlsclient import coapDtlsClient
3 from uuid import getnode as get_mac
4 import time
5
6 #Temperature sensor file
7 temp_file = '/sys/bus/w1/devices/28-0317030bf2ff/w1_slave'
8 #SENSOR          MAC_SENSOR          TEMP          FLUXO          GPS
        TEMPO
9 def mountPayload(sensorId, sensorMac, temp, flow, gps, timestamp):
10     payload = sensorId

```

```

11     payload += '/' + sensorMac
12     payload += '/' + temp
13     payload += '/' + flow
14     payload += '/' + gps
15     payload += '/' + timestamp
16     print(payload)
17     return payload
18
19
20     ciphers = "ECDHE+AESGCM"
21     clienteDtls = coapDtlsClient("35.231.29.66", 5684, ciphers).client
22     #temperatureSensor = TempSensor(temp_file);
23     try:
24         while True:
25             temperature = str(temperatureSensor.read_temp());
26             macAddress = str(get_mac())
27             timestamp = str(time.time())
28             payload = mountPayload("SENSOR_JOHANN", macAddress, temperature,
29                                   str(0), "NaMinhaCasa", timestamp)
30             tInicio = time.time()
31             response = clienteDtls.post("gpsflowtemp", payload)
32             tFim = time.time()
33             print("Post demorou %f ms na configuracao %s" % (tFim - tInicio,
34                                                             ciphers))
35             print response.pretty_print()
36             time.sleep(60)
37     except KeyboardInterrupt:
38         print "cliente stopped"
39         clienteDtls.stop()
40         print "Exiting..."

```

Listagem A.3 – tempsensor.py

```

1 import os
2 import time
3
4 #os.system('modprobe w1-gpio')
5 #os.system('modprobe w1-therm')
6 #temp_file = '/sys/bus/w1/devices/28-0317030bf2ff/w1_slave'
7
8 class TempSensor(object):
9     def __init__(self, temp_file):

```

```
10     super(TempSensor, self).__init__()
11     os.system('modprobe w1-gpio')
12     os.system('modprobe w1-therm')
13     self.temp_file = temp_file
14
15     def read_temp_file(self):
16         f = open(self.temp_file, 'r')
17         lines = f.readlines()
18         f.close()
19         return lines
20
21     def read_temp(self):
22         lines = self.read_temp_file()
23         while lines[0].strip()[-3:] != 'YES':
24             time.sleep(0.1)
25             lines = self.read_temp_file()
26         temp_output = lines[1].find('t=')
27         if temp_output == -1:
28             return -1
29         temp_string = lines[1].strip()[temp_output+2:]
30         temp_c = float(temp_string) / 1000.0
31         return temp_c
```

Listagem A.4 – coapdtlserver.py

```
1 # System
2 import socket
3 import random
4 import threading
5 import unittest
6 import tempfile
7 import os
8
9 # Dtls
10 import ssl
11 from dtls.wrapper import wrap_server, wrap_client
12
13 # CoAPthon
14 from coapthon import defines
15 from coapthon.server.coap import CoAP as CoAPServer
16 from coapthon.messages.option import Option
17 from coapthon.messages.request import Request
```

```

18 from coapthon.messages.response import Response
19 # Request handler (POST, GET, PUT, DELETE)
20 from coapdtlsrequesthandler import GpsFlowTempResource
21
22 # Logging
23 #from logging import basicConfig, DEBUG, getLogger, root, Filter
24 #basicConfig(level=DEBUG, format="%(asctime)s - %(threadName)-30s - %(name)s -
    %(levelname)s - %(message)s")
25
26 CA_CERT_VALID = """-----BEGIN CERTIFICATE-----
27 MIICczCCAXQCCQCwvSKaN4J3cTANBgkqhkiG9w0BAQUFADBKMQswCQYDVQQGEwJV
28 UzETMBEGA1UECBMKV2FzaGluZ3RvbjETMBEGA1UEChMKUmF5IENBIEluYzERMA8G
29 A1UEAxMIUmF5Q0FJbmMwHhcNMTQwMTE4MjEwMjUwWhcNMjEwMTE4MjEwMjUwWjBK
30 MQswCQYDVQQGEwJVUzETMBEGA1UECBMKV2FzaGluZ3RvbjETMBEGA1UEChMKUmF5
31 IENBIEluYzERMA8GA1UEAxMIUmF5Q0FJbmMwZ8wDQYJKoZIhvcNAQEBBQADgYOA
32 MIGJAoGBAN/UYXt4uq+YdTDnm7WPCu+OB50kJXWU3sSS+WAAhr3BHh7qa7UTiRXy
33 yGYysgvtwriETAZRckzd+hdb1NRUWXGJdRvtyx94nLpPpI8p4djBrJ5IMPqK5SgW
34 ZP4XTWs694VtUBAvHCX+Ly+t005Rw3NmqxY1MakooqU9t+wLOHOTAgMBAAEwDQYJ
35 KoZIhvcNAQEFBQADgYEANemjvYCJrTc/6im0DmDC6AW8KrLG0xj31HWpq1d09LG7
36 m1VFgbVtbcuCZgA78kxgw1vN6kBBLEsAJC8gkg++A0/w3a4oP+U9txAr9KRg6IGA
37 FiUohuWbjKBnQEpcEOECgrymooF3ayzke/vf3wcMYy153uC+H4t96Yc5T066c4o=
38 -----END CERTIFICATE-----
39 """
40
41 CA_CERT_WRONG = """-----BEGIN CERTIFICATE-----
42 MIIIE6jCCBF0gAwIBAgIDEIGKMAOGCSqGSIb3DQEBBQUAME4xCzAJBgNVBAYTA1VT
43 MRAwDgYDVQQKEwdFcXVpZmF4MS0wKwYDVQQLEyRfcXVpZmF4IFN1Y3VyZSBDZXJ0
44 aWZpY2F0ZSBDbXR0b3JpdHkwHhcNMTAwNDMwMDEwMDEwWhcNMTEwNDMwMDEwMDEw
45 WjCBjzEpMCCGA1UEBRMgMmc4YU81d0kxYktKMlpENTg4VXNMdkRlM2dUYmc4RFUx
46 CzAJBgNVBAYTA1VTMRMwEQYDVQQIEwPwYXpZmF4Ym1hMRIwEAYDVQQHEw1TdW5u
47 eXZhbGUxZDAsBgNVBAoTC11haG9vICBjbMUMRYwFAYDVQQDEw13d3cueWFob28u
48 Y29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA6ZM1jHckL8r1EKse
49 1riTTxyC3WvYQ5m34TlFK7dK4QFI/HPttKGqQm3aVB1Fqi0aiTxe4YQMbd++jnKt
50 djxcpi7sJlFxmZs4umr1eGo2KgTgSBAJyhxo23k+VpK1SprdPyM3yEfQVdV7JWC
51 4Y71CE2nE6+GbsIuhk/to+jJM07jXx/430jvo8vhNPL6GvWe/D60bbnxS72ynLSd
52 mLtalyk0vZEZiXbbFKGIaYYmCgh89FGVvBkUbGM/Wb5Voiz7ttQLLxKOYRj8Mdk
53 TZtzPkM9scIFG1naECPvCwx0NyMyxY3nF0djUKJ79twanmfCclX2Z0/rk1Cpi0uw
54 lrrrr/QIDAQABo4ICDjCCAgowDgYDVR0PAQH/BAQDAgTwMBOGA1UdDgQWBBSmrfKs
55 68m+dDUSf+S7xJrQ/FXAlza6BgNVHR8EMZAxMC+gLaArhilodHRwOi8vY3JsLmdl
56 b3RydXN0LmNvbS9jcmxzL3N1Y3VyZW5hLmNybDCCAVsGA1UdEQSCAVIwggF0gg13
57 d3cueWFob28uY29tggl5YWhvby5jb22CDHVzLn1haG9vLmNvbYIMa3IueWFob28u
58 Y29tgggx1ay55YWhvby5jb22CDG11Ln1haG9vLmNvbYIMZnIueWFob28uY29tgggxp

```

```
59 bi55YWhvby5jb22CDGNhLn1haG9vLmNvbYIMYnIueWFob28uY29tggxkZS55YWhv
60 by5jb22CDGVzLn1haG9vLmNvbYIMbXgueWFob28uY29tggxpdC55YWhvby5jb22C
61 DHNnLn1haG9vLmNvbYIMaWQueWFob28uY29tggxwaC55YWhvby5jb22CDHFjLn1h
62 aG9vLmNvbYIMdHcueWFob28uY29tggxoay55YWhvby5jb22CDGNuLn1haG9vLmNv
63 bYIMYXUueWFob28uY29tggxhci55YWhvby5jb22CDHZuLn1haG9vLmNvbTafBgNV
64 HSMEGDAWgBRI5mj5K9Ky1ddH2CMgEE8zmJCf1DAdBgNVHSUEFjAUBggrBgEFBQcD
65 AQYIKwYBBQUHAWIwDQYJKoZIhvcNAQEFBQADgYEAp9WOMtcDMM5T0yfpPecGv5QhH
66 RJZRzgeMPZitLksr1JxxicJrdgv82NWq1bw8aMurj47ijrtaTEWxaCQCy00yXodD
67 zoRJVnoYIvY1arYZf5zv9VZjN5IOHqUc39mNMe9XdZtbkWE+K6yVh60imKLbizna
68 inu9YTrN/4P/w6KzHho=
69 -----END CERTIFICATE-----
70 ""
71
72 KEY_CERT_VALID = ""-----BEGIN PRIVATE KEY-----
73 MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBANjL+g7MpTEB40Vo
74 2pxWbx33YwgXQ6QbnLg1QyKlrH6DEEotyDRWI/ZftvWbjGUhOzUGhQaLzF3ZNgdM
75 VkF5j0wCgRdwPon1ct5wJUg6GCWvfi4B/H1QrWg8JDaWoGuDcTqLh6KYfDdWT1WC
76 Bq3p0W14gVe3d12R8Bxu9PCK8jrvAgMBAAECgYAQFjqs5HSRiWFS4i/uj99Y6uV3
77 UTqcr8vWQ2WC6aY+EP2hc3o6n/W1L28FFJC7ZGImuiAe1zrH7/k5W2m/HAUM7M9p
78 oBcp7ZVMFU6R00cQWVKCpQRcPNHnn+tVJdRgiHRj9836/u2z3shBxDYgXJIR787V
79 SlBXkCcsi0Clem5ocQJBAPp/OtF4CpoaOCAnNN+rDjPNGcH571mpSZBMXZVAVCRq
80 vJDdH9SIcb19gkTocF1MUd7CJWbSHKxh49Hr+prBW8cCQQDdjH8EZ4CDYvoJbVX
81 iWfFbh61Pwv8uaj43HoHq4+51mhHvLx08a1AKMSgd2cg7yJYYIpTTAf21gqU3Yt9
82 wJeZakEA175e4u0o3vkLDs8xRFzGmbKg69SPA11+ap8YAZWaYwUVfVu2MHUHEZa5
83 GyxEB0B6p8pMBeE55WLMw8UHDMNeQJADEWRGjMnm1mAvFUKXFThrdV9oQ2C7nai
84 I1ai87X0+i4kDIUpsP21603ZJjx0K+DS+C4wuzhk4IkugNxck5SNUQJASxf8E4z5
85 W5rP2XXIohGpDyzI+criUYQ6340vKB9bPsCQ2QooQq1BH0wGA2fY82Kr95E8KhUo
86 zGoP1Dtpzgw0Qg==
87 -----END PRIVATE KEY-----
88 -----BEGIN CERTIFICATE-----
89 MIICDTCCAXYCCQCxc2uXBLZhDjANBgkqhkiG9w0BAQUFAADBKMqswCQYDVQQGEwJV
90 UzETMBEGA1UECBMKV2FzaGluZ3RvbjETMBEGA1UEChMKUmf5IENBIEluYzERMA8G
91 A1UEAxMIUmf5QOFJbmMwHhcNMTQwMTE4MjEwMjUwWhcNMjEwMTE4MjEwMjUwWjBM
92 MQswCQYDVQQGEwJVUzETMBEGA1UECBMKV2FzaGluZ3RvbjEUMBIGA1UEChMLUmf5
93 IFNydiBjbmMxejAQBgnVBAMTCVJheVNYdkluYzCBnzANBgkqhkiG9w0BAQEFAAOB
94 jQAwwYkCgYEA2Mv6Dsy1MQHjRwJanFZvHfdjCBdDpBucuDVDIqWsfomQSi3INFYj
95 91+29ZuMZSHTNqafBovMXdk2B0xwQXmPTAKBF3A+ifVy3nA1SDoYJa9+LgH8eVct
96 aDwkNpaga4Nx0ouHoph8N1ZOVYIGrek5bXiBV7d3XZHWHG708IryOu8CAwEAATAN
97 BgkqhkiG9w0BAQUFAAOBgQBwOXUTYzfiIOFi9g4GuyWD2hjET3NtrT4Ccu+Jiivy
98 EvwhzHtVGAPhrV+VCL8sS9uSOZlmfK/ZVraDiFGpJLDMvPP5y5fwq5VGrFuZispG
99 X6bTbQ2AIKzGGXxhwPqD8F7su7bmZDnZFRMRk2Bh16rv0mtzx9yHtqC5YJZ2a3JK
100 2g==
```

```

101 -----END CERTIFICATE-----
102 """
103
104
105 class coapDtlsServer(object):
106
107     def __init__(self, host, port):
108         self.host_address = (host, port)
109         #self.current_mid = random.randint(1, 1000)
110         #self.server_mid = random.randint(1000, 2000)
111         #self.server_read_timeout = 0.1
112
113         self.pem = self._setUpPems()
114
115         # Set up a server side DTLS socket
116         _sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
117         _sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
118         _sock = wrap_server(_sock,
119                             keyfile=self.pem['SERVER_KEY'],
120                             certfile=self.pem['SERVER_KEY'],
121                             ca_certs=self.pem['CA_CERT'])
122         _sock.bind(self.host_address)
123         _sock.listen(0)
124
125         # Connect the CoAP server to the newly created socket
126         self.server = CoAPServer(self.host_address,
127                                 starting_mid=None,
128                                 sock=_sock,
129                                 cb_ignore_listen_exception=self._cb_ignore_listen_exception)
130         self.server.add_resource('gpsflowtemp/', GpsFlowTempResource())
131
132
133     def _setUpPems(self):
134         def _createPemFile(fname, content):
135             with open(fname, mode='w') as f:
136                 f.write(content)
137             f.close()
138             return fname
139
140         self._tmp_dir = tempfile.mkdtemp()
141
142         pem = dict()

```



```
143 pem['SERVER_KEY'] = _createPemFile(os.path.join(self._tmp_dir,
144                                     'serverkey.pem'), KEY_CERT_VALID)
145 pem['CA_CERT'] = _createPemFile(os.path.join(self._tmp_dir,
146                                             'ca_cert.pem'), CA_CERT_VALID)
147 pem['CA_CERT_WRONG'] = _createPemFile(os.path.join(self._tmp_dir,
148                                                 'ca_cert_wrong.pem'), CA_CERT_WRONG)
149
150 return pem
151
152 def _cb_ignore_listen_exception(self, exception, server):
153     """
154     In the CoAP server listen method, different exceptions can arise from
155     the DTLS stack. Depending on the type of exception, a
156     continuation might not be possible, or a logging might be desirable.
157     With this callback both needs can be satisfied.
158
159     :param exception: What happened inside the DTLS stack
160     :param server: Reference to the running CoAP server
161     :return: True if further processing should be done, False processing
162             should be stopped
163     """
164     if isinstance(exception, ssl.SSLError):
165         # A client which couldn't verify the server tried to connect,
166         # continue but log the event
167         if exception.errqueue[-1][0] == ssl.ERR_TLSV1_ALERT_UNKNOWN_CA:
168             #print("Ignoring ERR_TLSV1_ALERT_UNKNOWN_CA from client %s" %
169                   #      ('unknown' if not hasattr(exception, 'peer') else
170                       str(exception.peer)))
171             return True
172         # ... and more ...
173     return False
174
175 def main():
176     #cloud do loffi
177     server = coapDtlsServer("0.0.0.0", 5684).server;
178     try:
179         server.listen(10)
180     except KeyboardInterrupt:
181         print "Server Shutdown"
182         server.close()
183         print "Exiting..."
184
```

```

177 if __name__ == '__main__':
178     main()

```

Listagem A.5 – coapdtlsrequesthandler.py

```

1 from coapthon.server.coap import CoAP
2 from coapthon.resources.resource import Resource
3 from dbhandler import DataBase
4
5 class GpsFlowTempResource(Resource):
6     def __init__(self, name="gpsflowtemp", coap_server=None):
7         super(GpsFlowTempResource, self).__init__(name, coap_server,
8             visible=True,
9
10                observable=True, allow_children=True)
11
12         self.payload = None
13         self.dbhandler = DataBase()
14         self.dbhandler.connect_db()
15
16     def render_GET(self, request):
17         return self
18
19     def render_PUT(self, request):
20         return self
21
22     def render_POST(self, request):
23         res = self.init_resource(request, GpsFlowTempResource())
24         res.location_query = request.uri_query
25         res.payload = request.payload
26         print("\n"+res.payload)
27         self.dbhandler.on_message(request.payload)
28         return res
29
30     def render_DELETE(self, request):
31         return True

```

Listagem A.6 – dbhandler.py

```

1  #!/usr/bin/env python 1
2  # -*- coding: utf-8 -*-
3
4  import sys
5  import MySQLdb
6

```

```
7 class DataBase(object):
8     """docstring for ClassName"""
9     def __init__(self):
10         self.cursor = None;
11         self.db = None;
12
13     def connect_db(self):
14         try:
15             self.db = MySQLdb.connect("localhost","usuario","senha","nomeBanco")
16             self.cursor = self.db.cursor()
17         except:
18             print("No se pode conectar ao servidor de banco de dados")
19             print("TERMINANDO...")
20             sys.exit()
21
22     def on_message(self, dados):
23         lista = dados.split("/")
24         sql = 'INSERT INTO
25             IoT_MQTT.dados(sensor,mac_sensor,temp,fluxo_agua,gps,data_ins)
26             VALUES (''+ lista[0]+'',''+lista[1]+'', ''+lista[2]+'',
27             ''+lista[3]+'',''+str(lista[4])+'', CURRENT_TIMESTAMP);'
28         try:
29             self.cursor.execute(sql)
30             self.db.commit()
31         except:
32             self.db.rollback()
33             print("Falha ao gravar no Banco de Dados!")
```

APÊNDICE B – Artigo no formato SBC

Nas páginas seguintes está o artigo referente a este trabalho de conclusão de curso, respeitando o formato da SBC (Sociedade Brasileira de Computação).

Desenvolvimento de uma Aplicação com dispositivo IoT usando Protocolos DTLS e CoAP

Johann Westphall¹

¹Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

johannwestphall@gmail.com

Abstract. *Some specific protocols were developed aiming to help the communication of restricted small devices, through the Internet, without overloading them. CoAP (Constrained Application Protocol) and DTLS (Datagram Transport Layer Security) represent two of these protocols.*

Protocols like the above help to secure the communication in many problems involving restricted IoT devices. This work regards secure communication in IoT devices, which collect data from plantations and the use of pesticides.

Resumo. *Certos protocolos foram criados para facilitar a comunicação de dispositivos limitados com a Internet das Coisas, com o objetivo de não sobrecarregá-los. Exemplos de protocolos são: CoAP (Constrained Application Protocol) e DTLS (Datagram Transport Layer Security).*

Protocolos assim podem ser utilizadas para auxiliar a comunicação segura nos mais diversos problemas envolvendo dispositivos de IoT. Este trabalho trata de comunicação segura utilizando dispositivos IoT na coleta de dados referentes a uma plantação e o uso de pesticidas.

1. Introdução

O desenvolvimento computacional facilitou a diversificação e construção de novos dispositivos, tais como sensores e microcontroladores. Sensores são capazes de medir intensidade luminosa, presença humana, velocidade, eletricidade, nível de líquidos, força e outros fatores, e com isso auxiliam aplicações [Kamaludin and Ismail 2017]. Todavia, sensores e microcontroladores possuem limitações de alimentação energética, de processamento e de armazenamento, exigindo protocolos eficientes e eficazes [Bormann 2016] [Bormann et al. 2012].

A Internet das Coisas (*IoT - Internet of Things*) conecta esses objetos físicos compostos por elementos eletrônicos, software e sensores com a infraestrutura de rede como a Internet [Zhou et al. 2017].

Muitos dispositivos IoT trazem grandes facilidades às nossas vidas e podem ser utilizados em contextos de grande importância, nos quais o sigilo é fundamental. Além disso, sensores ficam em lugares de fácil acesso e alteração. Assim, o desenvolvimento de aplicações que usam mecanismos de segurança são fundamentais, considerando as limitações dos dispositivos.

Este artigo apresenta o desenvolvimento de uma aplicação com dispositivos IoT usando o protocolo de segurança DTLS (*Datagram Transport Layer Security*) e o protocolo para aplicações restritas chamado CoAP (*Constrained Application Protocol*).

O restante do artigo está organizado da seguinte maneira: na seção 2 são descritos os trabalhos relacionados da literatura; a seção 3 apresenta os conceitos básicos de IoT (*IoT - Internet of Things*), CoAP e DTLS; na seção 4 é descrito o desenvolvimento da aplicação proposta para prover segurança ao ambiente proposto e na última seção são descritas as conclusões do trabalho.

2. Trabalhos relacionados

Dois trabalhos de pesquisa são descritos nesta seção: [da Porciúncula et al. 2018] e [Banerjee et al. 2018].

O trabalho descrito em [da Porciúncula et al. 2018] trata do uso do CoAP como protocolo de comunicação em dispositivos restritos, ou seja, com pouca capacidade de processamento, energia e memória. Esses dispositivos são classificados em três categorias, de acordo com sua capacidade de armazenamento de dados e de código.

O experimento dos autores envolve um Arduino UNO conectado à Internet por um cabo Ethernet, usando bibliotecas disponíveis para Ethernet em Arduino, para se comunicar com um servidor CoAP instalado em um notebook na mesma rede. O Arduino utilizou a biblioteca *microcoap* e o servidor usou a biblioteca *CoAPthon*. Os autores comprovaram a comunicação usando o *WireShark* para capturar pacotes do protocolo.

Os autores do trabalho [Banerjee et al. 2018] abordam o problema de segurança em dispositivos restritos com um hardware específico que implementa alguns modos do protocolo DTLS. A justificativa, segundo eles, é que o DTLS exclusivamente implementado em software consome grande espaço em memória e é mais lento, caso comparado a uma implementação em hardware. Só o processamento de criptografia de curvas elípticas (ECC), ocupa 99% do tempo da fase de handshake do DTLS, assim, é vantajoso um acelerador criptográfico para melhorar o desempenho do protocolo.

3. Conceitos básicos

Nesta seção são apresentados os conceitos básicos para o entendimento da aplicação desenvolvida: IoT, CoAP e DTLS.

3.1. IoT - Internet of Things

IoT é a estrutura que viabiliza a comunicação entre grandes servidores e computadores com pequenos e restritos microcontroladores. Engloba vários tipos de redes como: Redes de Sensores Sem Fio (WSN - *Wireless Sensor Network*), Máquina-a-Máquina (M2M - *Machine-to-Machine*), Redes Sem Fio de Baixo Consumo (LoWPAN - *Low-Power Wireless Personal Area Networks*) e Identificação com Rádio-Frequência (RFID - *Radio-Frequency Identification*).

Generalizando, IoT materializa a visão de rede na qual sensores terão a capacidade de se comunicar com outros dispositivos usando os protocolos da Internet [Granjal et al. 2015]. Ou seja, IoT é composta por eletrônicos embarcados, softwares e sensores, que facilitam a integração entre o mundo físico e as redes de computadores, trazendo benefícios [Zhou et al. 2017].

3.2. CoAP - Constrained Application Protocol

O HTTP (*HyperText Transfer Protocol*) passou por mais de uma década de crescimento organizado, acumulando cada vez mais implementações, o que supera a capacidade de ser usado em dispositivos pequenos [Bormann et al. 2012].

O CoAP é um protocolo de transferência de informações da camada de aplicação, com o mesmo conjunto de operações básicas se comparado ao HTTP, porém consideravelmente mais simples, consumidor de menos memória, processamento e, consequentemente, energia [Bormann et al. 2012]. O principal fator responsável pela redução da complexidade é o uso do UDP (*User Datagram Protocol*) ao invés do TCP (*Transmission Control Protocol*). O CoAP trata o problema de perda de pacotes, não solucionado pelo UDP, adicionando uma camada à mensagem para detecção e retransmissão dos pacotes. Por esse motivo, um cabeçalho CoAP tem apenas quatro bytes somados a dois bytes para opções e a resposta da requisição ocupa apenas um byte. Assim, uma requisição geralmente usa de 10 a 20 bytes.

3.3. DTLS - Datagram Transport Layer Security

Um dos protocolos de segurança computacional mais usados é o TLS (*Transport Layer Security*), o qual possibilita uma série de configurações como algoritmo de troca de chaves, algoritmo de criptografia assimétrica algoritmo de criptografia simétrica, modo de operação e algoritmo de hash [Dierks 2008]. Contém também envios de certificados X.509 relacionados a criptografia assimétrica e além disso, o TLS troca informações pela rede usando o TCP, de forma a obter garantia de recebimento. Como já citado, o TCP é evitado em dispositivos pequenos com pouca capacidade energética e de processamento, devido ao seu custo e por isso nesse contexto há preferência pelo UDP, causando uma incompatibilidade com o TLS.

Em muitos casos, a maneira mais desejável para garantir segurança entre cliente e servidor seria usando o TLS, todavia o uso do UDP impossibilita o uso do TLS, por isso o DTLS (TLS para datagramas) foi projetado para ser o mais similar ao TLS possível, minimizando novas invenções de seguranças e possibilitando aproveitamento de código e estrutura do TLS [Rescorla 2012].

A Figura 1 mostra o estabelecimento do *handshake* DTLS, o qual consiste em uma sequência de troca de mensagens, de forma que, ao fim dessa troca, as configurações de segurança entre cliente e servidor estão estabelecidas. Assim como no TLS, no DTLS o *handshake* serve estabelecer os algoritmo de troca de chaves, algoritmo de autenticação, algoritmo de criptografia simétrica e algoritmo de hash do HMAC. Diferentemente do TLS, no DTLS as mensagens são trocadas no *handshake* em *Flight*. Em caso de uma mensagem perdida, todas as mensagens relacionadas ao *flight* dessa mensagem perdida são retransmitidas.

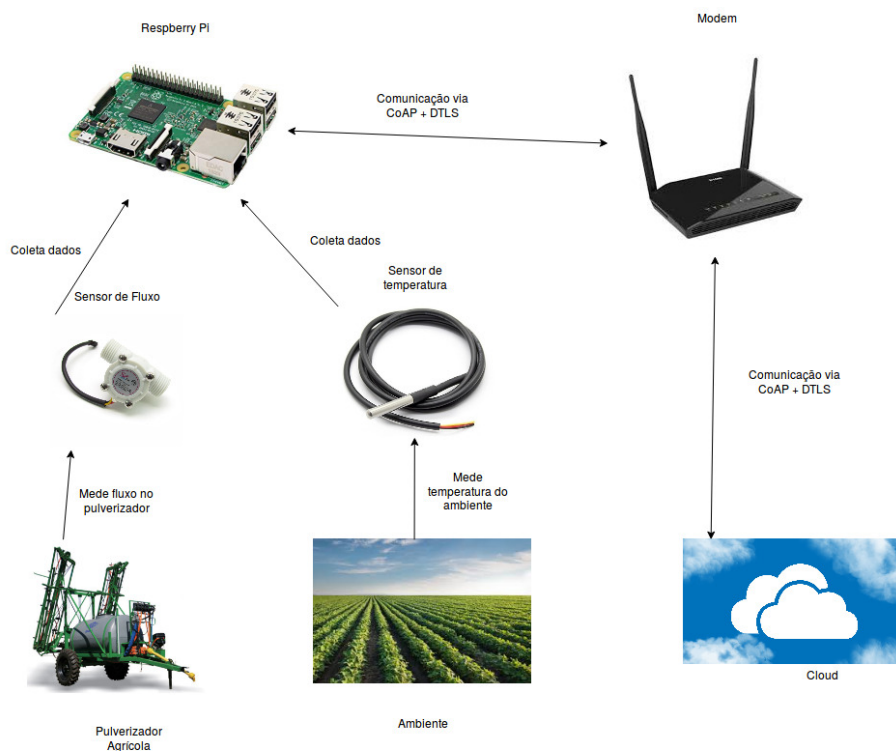


Figura 2. Uso de IoT em aplicação agrícola

Após a coleta dos dados, a Raspberry Pi envia essas informações para a Cloud através de um gateway (modem representado na Figura 2) de maneira segura, utilizando o CoAP como protocolo da camada de aplicação e o DTLS como protocolo da camada de segurança. A Raspberry Pi se comporta como um cliente CoAP, enquanto a Cloud se comporta como servidor, pois basta a Raspberry Pi conhecer o endereço do servidor e enviar dados. A Cloud, como servidor, é responsável por processar e armazenar os dados obtidos.

4.3. CoAP e DTLS: biblioteca CoAPthon

CoAPthon é uma biblioteca em python para o protocolo CoAP [Tanganelli 2018, Tanganelli et al. 2015].

Como o próprio autor da biblioteca informa [Tanganelli et al. 2015], CoAPthon é uma implementação de CoAP usando a linguagem Python que possui as funcionalidades descritas no documento RFC 7252 [Bormann 2014] do CoAP. Assim como a biblioteca ESP-CoAP, CoAPthon implementa as operações básicas do CoAP, tanto no papel de cliente quanto no papel de servidor, todavia supera a ESP-CoAP ao implementar a função de, por exemplo, utilizar *proxy* entre CoAP e HTTP.

Outra vantagem da CoAPthon é a presença de uma implementação do DTLS para utilização conjunta ao CoAP caso desejado pelo usuário da biblioteca. Durante o processo de pesquisa de implementações envolvendo CoAP com DTLS, foi difícil encontrar uma biblioteca que de fato permitia o CoAP operar com o DTLS.

4.4. Implementação

O autor da biblioteca CoAPthon disponibiliza no seu repositório do GitHub [Tanganelli 2018] alguns testes envolvendo CoAP em conjunto com o DTLS no arquivo "test_secure.py". Por se tratar de um teste, tanto cliente e servidor foram instanciados no mesmo arquivo de código. Tendo em vista esse fato, foram feitas adaptações e modificações no código existente, sendo que as principais tarefas executadas no desenvolvimento foram:

- Criação do código da parte cliente e da parte servidor envolvendo a união dos protocolos CoAP e DTLS, com base nos testes realizados pelo autor da biblioteca.
- Desenvolvimento dos códigos para leitura de informações dos sensores de temperatura.
- Desenvolvimento do código para unir a leitura de sensores e enviar os dados utilizando CoAP com DTLS.
- Desenvolvimento do código para o servidor salvar as informações recebidas em um banco de dados.
- Desenvolvimento do código para lidar com as requisições enviadas ao servidor (POST, GET, DELETE e PUT).
- Desenvolvimento do código para unir operações do banco de dados, criação do servidor CoAP com DTLS e o tratamento de requisições.
- Implantação dos códigos relacionados ao servidor em uma Cloud da Google.
- Implantação dos códigos relacionados ao cliente na Raspberry Pi 3.

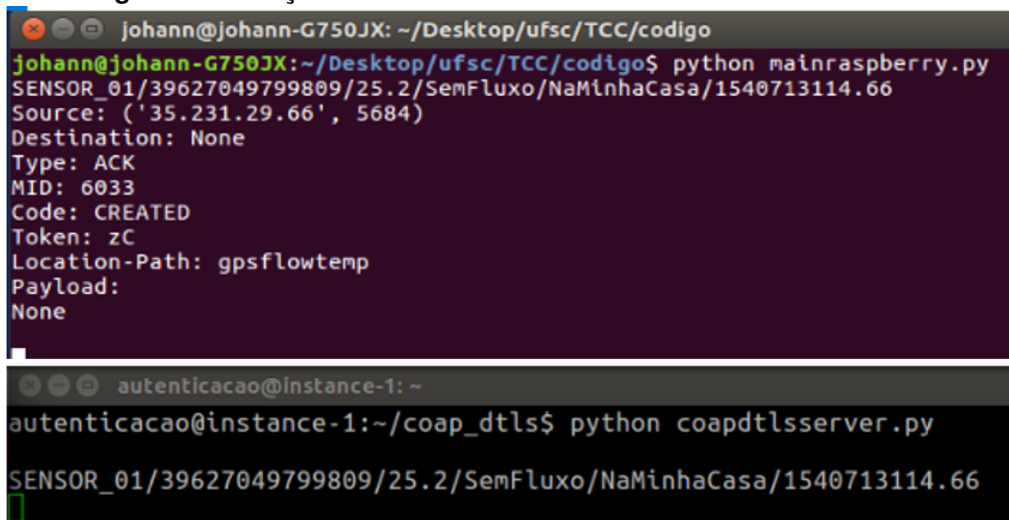
4.5. Testes

Após a implantação do servidor em uma cloud da Google, foram realizados testes para verificação da corretude dos códigos desenvolvidos.

O primeiro teste foi realizado com um notebook executando o código do cliente, que é executado pela Raspberry Pi, e o servidor na cloud da Google. O objetivo do primeiro teste foi verificar que a string contendo as informações enviadas pelo cliente são decifradas corretamente no servidor (Figura 3). Na primeira linha do terminal mostrado na parte superior da Figura 3 está a string enviada do cliente ao servidor, com informações em sequência separadas por "/".

A primeira informação, "SENSOR_01" representa o nome da Raspberry Pi que está enviando os dados. Em seguida, o número "39627049799809" representa o MAC address da Raspberry Pi. As próximas três informações representam respectivamente a temperatura em Grau Celsius, o fluxo e a localização lidos pelos sensores. Por último, é enviado um timestamp considerando o relógio do sistema operacional da Raspberry Pi. Nas linhas seguintes do mesmo terminal, estão os campos do cabeçalho do CoAP referentes ao *Acknowledgement* enviado pelo servidor ao receber as informações da primeira linha.

Figura 3. Interação entre cliente e servidor utilizando CoAP e DTLS



```
johann@johann-G750JX: ~/Desktop/ufsc/TCC/codigo
johann@johann-G750JX:~/Desktop/ufsc/TCC/codigo$ python mainraspberry.py
SENSOR_01/39627049799809/25.2/SemFluxo/NaMinhaCasa/1540713114.66
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 6033
Code: CREATED
Token: zC
Location-Path: gpsflowtemp
Payload:
None

autenticacao@instance-1: ~
autenticacao@instance-1:~/coap_dtls$ python coapdtlsserver.py

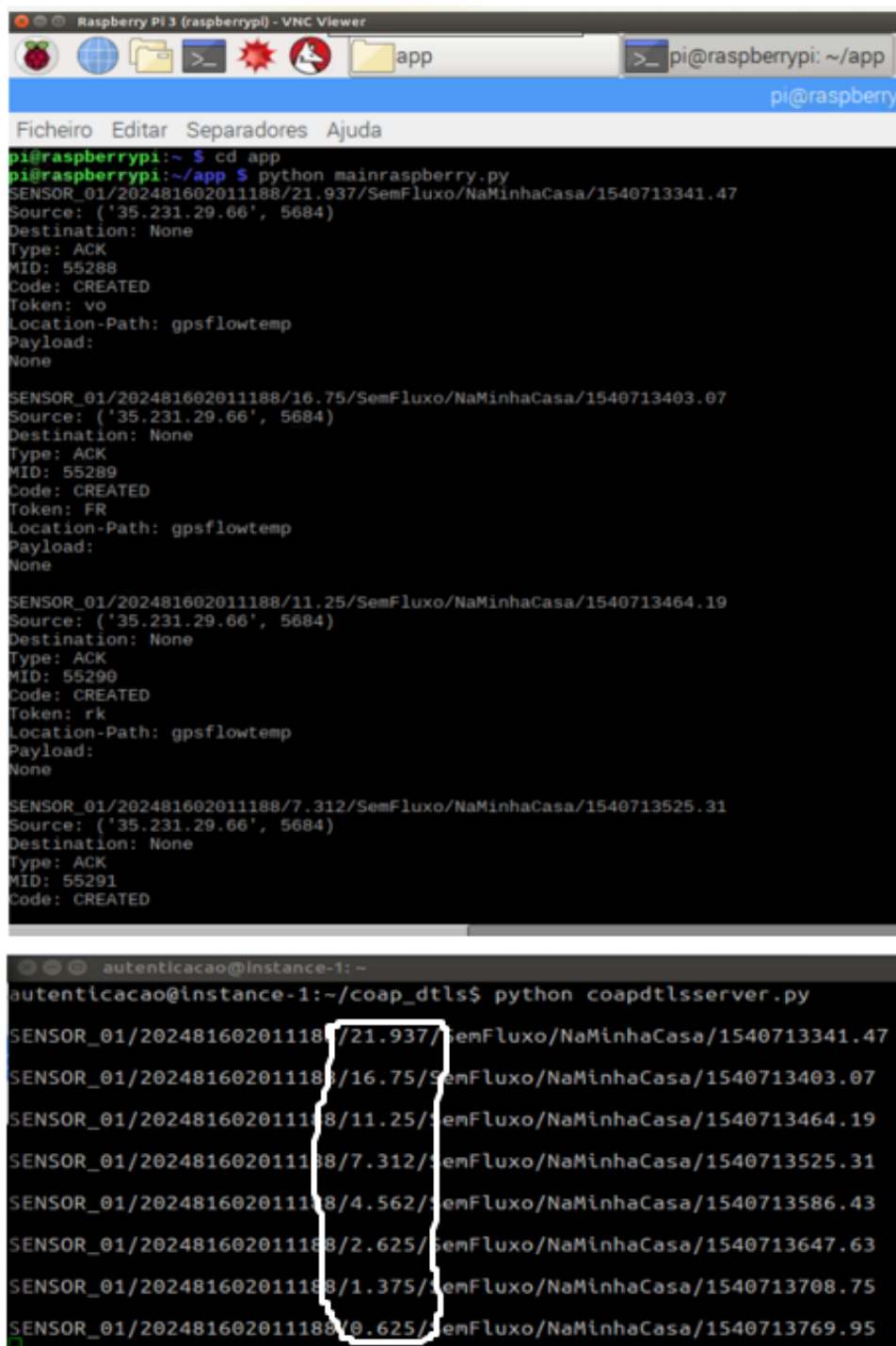
SENSOR_01/39627049799809/25.2/SemFluxo/NaMinhaCasa/1540713114.66
```

Fonte: Elaborada pelo autor

A Figura 3 mostra na janela superior a execução do cliente, executado em um notebook, com uma temperatura fictícia de 25.2°C e na janela inferior a execução do servidor. Nota-se que a mensagem enviada pelo cliente é recebida com sucesso no servidor e o ACK relativo ao protocolo CoAP é recebido pelo cliente como indicação de sucesso na comunicação.

A Figura 4 mostra o segundo teste, envolvendo o código desenvolvido em execução na Raspberry Pi, para verificar o funcionamento do código de leitura da temperatura e identificar possíveis ausências de bibliotecas python necessárias. A fim de controlar a Raspberry Pi remotamente, foi utilizado o programa VNC Viewer.

Figura 4. Interação entre Raspberry Pi e servidor utilizando CoAP e DTLS



```
Raspberry Pi 3 (raspberrypi) - VNC Viewer
pi@raspberrypi: ~/app
Ficheiro Editar Separadores Ajuda
pi@raspberrypi:~ $ cd app
pi@raspberrypi:~/app $ python mainraspberrypi.py
SENSOR_01/202481602011188/21.937/SemFluxo/NaMinhaCasa/1540713341.47
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 55288
Code: CREATED
Token: vo
Location-Path: gpsflowtemp
Payload:
None

SENSOR_01/202481602011188/16.75/SemFluxo/NaMinhaCasa/1540713403.07
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 55289
Code: CREATED
Token: FR
Location-Path: gpsflowtemp
Payload:
None

SENSOR_01/202481602011188/11.25/SemFluxo/NaMinhaCasa/1540713464.19
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 55290
Code: CREATED
Token: rk
Location-Path: gpsflowtemp
Payload:
None

SENSOR_01/202481602011188/7.312/SemFluxo/NaMinhaCasa/1540713525.31
Source: ('35.231.29.66', 5684)
Destination: None
Type: ACK
MID: 55291
Code: CREATED

autenticacao@instance-1: ~
autenticacao@instance-1:~/coap_dtls$ python coapdtlsserver.py
SENSOR_01/202481602011188/21.937/SemFluxo/NaMinhaCasa/1540713341.47
SENSOR_01/202481602011188/16.75/SemFluxo/NaMinhaCasa/1540713403.07
SENSOR_01/202481602011188/11.25/SemFluxo/NaMinhaCasa/1540713464.19
SENSOR_01/202481602011188/7.312/SemFluxo/NaMinhaCasa/1540713525.31
SENSOR_01/202481602011188/4.562/SemFluxo/NaMinhaCasa/1540713586.43
SENSOR_01/202481602011188/2.625/SemFluxo/NaMinhaCasa/1540713647.63
SENSOR_01/202481602011188/1.375/SemFluxo/NaMinhaCasa/1540713708.75
SENSOR_01/202481602011188/0.625/SemFluxo/NaMinhaCasa/1540713769.95
```

Fonte: Elaborada pelo autor

O sensor de temperatura foi colocado sobre uma bolsa de gelo e, como mostra a Figura 4, os envios da Raspberry Pi ao servidor mostram a queda de temperatura.

5. Conclusão

Com o aumento do uso de dispositivos IoT, o atual trabalho traz contribuições para o ambiente de IoT de diversas maneiras. Primeiramente, o trabalho contém um estudo sobre dois protocolos relevantes, CoAP e DTLS, que podem facilitar a comunicação e a segurança dos dispositivos IoT.

Além do estudo teórico, o trabalho apresenta uma solução prática de IoT utilizando a combinação CoAP e DTLS, o que ainda não se encontra em artigos científicos e trabalhos acadêmicos com frequência. Ainda no escopo do desenvolvimento, a biblioteca CoAPthon possuía apenas um teste envolvendo DTLS, com códigos referentes ao cliente e ao servidor acoplados em um arquivo. Este trabalho demonstrou como criar em classes separadas o cliente e o servidor, utilizando CoAP e DTLS a partir da CoAPthon, com informações pertinentes de cada um.

O conteúdo deste trabalho auxilia também futuros trabalhos que utilizem a CoAPthon, já que a biblioteca não possui documentação completa. Neste trabalho foi desenvolvido um exemplo abordando também aspectos de configuração de segurança.

Trabalhos futuros podem tratar de aspectos ou realizar análises não cobertos neste trabalho, como por exemplo a integração de hardware e software do sistema desenvolvido neste trabalho com os sensores de fluxo e de localização. Também seria interessante um trabalho que adaptasse os códigos utilizados para dispositivos mais limitados do que uma Raspberry Pi, afinal, eles são o alvo principal do desenvolvimento de protocolos como DTLS e CoAP. Ainda no contexto de dispositivos limitados, pode ser realizado um estudo acerca da otimização entre a escolha de algoritmos de segurança e o consumo energético, espaço e tempo de processamento necessários.

Referências

- Banerjee, U., Juvekar, C., Wright, A., Arvind, and Chandrakasan, A. P. (2018). An energy-efficient reconfigurable dtls cryptographic engine for end-to-end security in iot applications. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 42–44.
- Bormann, C. (2014). The constrained application protocol (coap). Acessado em 3 de outubro de 2018.
- Bormann, C. (2016). Block-wise transfers in the constrained application protocol (coap). Acessado em 16 de Junho de 2018.
- Bormann, C., Castellani, A. P., and Shelby, Z. (2012). Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67.
- da Porciúncula, C. B., Beskow, S., Marcon, D. S., and Nobre, J. C. (2018). Constrained application protocol (coap) no arduino uno r3: Uma análise prática. volume 5.
- Dierks, T. (2008). The transport layer security (tls) protocol version 1.2. Acessado em 23 de junho de 2018.
- Granjal, J., Monteiro, E., and Silva, J. S. (2015). Security for the internet of things: A survey of existing protocols and open research issues. *IEEE Communications Surveys Tutorials*, 17(3):1294–1312.

- Kamaludin, K. H. and Ismail, W. (2017). Water quality monitoring with internet of things (iot). In *2017 IEEE Conference on Systems, Process and Control (ICSPC)*, pages 18–23.
- Rescorla, E. (2012). Datagram transport layer security version 1.2. Acessado em 19 de Junho de 2018.
- Tanganelli, G. (2018). Coapthon is a python library to the coap protocol aligned with the rfc. Acessado em 16 de outubro de 2018.
- Tanganelli, G., Vallati, C., and Mingozi, E. (2015). Coapthon: Easy development of coap-based iot applications with python. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 63–68.
- Zhou, J., Cao, Z., Dong, X., and Vasilakos, A. V. (2017). Security and privacy for cloud-based iot: Challenges. *IEEE Communications Magazine*, 55(1):26–33.